

مقدمه ای بر سی شارپ : قسمت- ۱

مقدمه :

در طی سلسله مقالاتی می خواهیم با C# بیشتر آشنا شویم. فرض این مقالات بر این است که آشنایی مختصری با زبانهای برنامه نویسی دارید ، هر چند کار ما تقریباً از صفر شروع می شود و هدف آن سادگی هر چه بیشتر است.

C# از دو زبان C++ و Java متولد شده است! حاوی بسیاری از جنبه های C++ می باشد اما ویژگی های شیء گرایی خودش را از جاوا به ارث برده است.

C# اگرچه از C++ گرفته شده است اما یک زبان "خالص" شیء گرا (Object oriented) می باشد. هر دو زبان یاد شده جزو زبانهای هیبرید محسوب می شوند اما طراحان C# این مورد را به اندازه ی C++ مهم تلقی نکرده اند. یک زبان هیبرید اجازه ی برنامه نویسی با شیوه های مختلف را میسر می کند. دلیل اینکه C++ هیبرید است ، این است که قرار بوده تا با زبان C سازگار باشد و همین امر سبب گردیده تا بعضی از جنبه های C++ بسیار پیچیده شوند.

زبان سی شارپ فرض اش بر این است که شما می خواهید تنها برنامه نویسی شیء گرا انجام دهید و همانند C++ مخلوطی از برنامه نویسی رویه ای (Procedural) و شیء گرا را نمی خواهید به پایان برسانید. بنابراین باید طرز فکر خودتان را با دنیای شیء گرایی تطبیق دهید. در ادامه خواهیم دید که در سی شارپ هر چیزی شیء است حتی یک برنامه ی سی شارپ.

برنامه ی اول :

Visual studio.net را اجرا کنید و سپس در صفحه ی ظاهر شده New Project را برگزینید. حالا از گزینه ی Visual C# projects قسمت Console applications را انتخاب نمایید. نامی دلخواه همانند ex01 را وارد نموده و سپس Ok نمایید. کد زیر به صورت خودکار برای شما تولید خواهد شد:

```
using System;

namespace ex01
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            //
            // TODO: Add code to start application here
            //
        }
    }
}
```

اگر یک سری از مفاهیم آنرا متوجه نمی شوید اصلاً مهم نیست! در مقالات آتی تمام این موارد مفصل توضیح داده خواهند شد.

متد استاندارد Main در اینجا قسمتی است که عملیات اصلی برنامه در حالت Console (شبیه به برنامه های تحت داس اما ۳۲ بیتی) در آن انجام می شود.
بدون متد Main برنامه های سی شارپ قادر به اجرا نخواهند بود. نوع آن در اینجا void تعریف شده است یعنی این متد خروجی ندارد. حتی اگر برنامه های استاندارد ویندوز را هم بخواهید با C# بنویسید بازهم متد Main حضور خواهد داشت ، هر چند به صورت خودکار ویژوال استودیو آنرا تولید می کند.

طریقه ی نوشتن توضیحات (Comments) در سی شارپ همانند C++ می باشد یعنی :

```
/* any comments */
```

و یا

```
// any comments
```

و تنها برنامه نویس برای نوشتن توضیحاتی در مورد کدهای خود از آنها استفاده می کند و در خروجی برنامه ظاهر نمی شوند.

فعلاً برای پایان قسمت اول از شیء Console و متد WriteLine آن برای نمایش یک جمله ی ساده استفاده می کنیم. راجع به متدها ، متغیرها و غیره در آینده بیشتر صحبت می کنیم.
در آخر برنامه ی ما چیزی شبیه به عبارت زیر می باشد:

```
using System;

namespace ex01
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            Console.WriteLine("Hello C#!");
        }
    }
}
```

دکمه ی F5 را فشار دهید تا برنامه اجرا شود.

مقدمه ای بر سی شارپ : قسمت- ۲

تعریف متغیرها در سی شارپ:

سی شارپ عناصری را که بکار می گیرد همانند اعداد و کاراکترها ، به صورت نوع ها (Types) طبقه بندی می کند. این انواع شامل موارد زیر می شوند :
نوع های پایه ایی از پیش تعریف شده مانند اعداد و غیره.
نوع های تعریف شده توسط کاربر که شامل STRUCT ها و ENUM ها می شوند.

نحوه ی تعریف متغیرها از نوع های پایه ایی از پیش تعریف شده :

همانطور که می دانید از متغیرها برای نگهداری اطلاعات استفاده می شود. در سی شارپ ابتدا نوع متغیر و سپس نام متغیر و در آخر یک سمی کولون بکار برده می شود. برای مثال :

```
int a;
```

که در اینجا متغیر a بعنوان يك متغیر حاوي اعداد صحیح تعریف شده است. نکته ي مهمي که در اینجا حائز اهمیت است ، مقدار دهی اولیه ي متغیرها می باشد. در غیر اینصورت کامپایلر سی شارپ برنامه را بایک خطا متوقف می کند. دلیل این امر هم این است که از استفاده از متغیرهای بدون مقدار در طول برنامه جلوگیری شود تا میزان خطاهای در حین اجرا کاهش یابد.

نوع های داده ای پایه ی زیر در در سی شارپ به صورت پیش فرض مهیا هستند:

object : نوعی است نامحدود که می تواند تمام انواع دیگر را نیز شامل شود. مثال :

object = null;

string : رشته ؛ در اینجا يك رشته توالی کاراکترهای یونیکد می باشد. مثال :

string s= "hello";

sbyte : نوع داده ایی صحیح ۸ بیتی علامت دار.

byte : نوع داده ایی صحیح ۸ بیتی بدون علامت. مثال :

sbyte val = 12;

short : نوع داده ایی صحیح ۱۶ بیتی علامت دار.

ushort : نوع داده ایی صحیح ۱۶ بیتی بدون علامت. مثال :

short val = 12;

int : نوع داده ایی صحیح ۳۲ بیتی علامت دار.

uint : نوع داده ایی صحیح ۳۲ بیتی بدون علامت. مثال :

int val = 12;

long : نوع داده ایی صحیح ۶۴ بیتی علامت دار.

ulong : نوع داده ایی صحیح ۶۴ بیتی بدون علامت. مثال :

Long val1 = 12; long val2 = 34L;

کلا در اینجا u به معنای unsigned است.

float : نوع اعشاری با single precision .

double : نوع اعشاری با double precision . مثال :

float val = 1.23f;

bool : نوع داده ایی Boolean که می تواند true و یا false باشد. مثال :

Bool val = true;

char : کاراکتر، در اینجا char يك کاراکتر یونیکد است.

char val = 'h';

به نحوه ی تعریف کاراکتر ها و همچنین رشته ها در سی شارپ دقت کنید.

decimal : نوع داده ایی دسیمال با 28 رقم معنی دار.

```
decimal val = 1.23M;
```

يك نکته :

- بهتر است هنگام تعريف يك متغير ، نامي با مسما براي آن انتخاب شود تا در هنگام كار خواندن كد ساده تر گردد. همچنين رسم شده است كه نوع متغير را به صورت خلاصه به نام متغير اضافه مي كنند. براي مثال بجاي FirstName بهتر است بنويسيم strFirstName . به اين نوع نگارش Hungarian notation مي گويند.
- تمام نوع هاي پيش فرض تعريف شده در سي شارپ شيء هستند. در آينده بيشتر در اين مورد صحبت خواهيم كرد.

مثال اين قسمت :

يك برنامه ي console جديد در را VS.NET باز كنيد. نام آنرا در ابتدا ex02 انتخاب نماييد. در اينجا مي خواهيم دو متغير رشته اي و صحيح را تعريف و سپس در خروجي نمايش دهيم.

كد نهايي به صورت زير مي باشد:

```
using System;

namespace ex02
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            int intVar1 = 0;

            int intVar2;
            intVar2=1;

            int intV3=15 , intV4 = 12;

            string strText1 = "abcd";

            Console.WriteLine(
                "The value for variables are : \n intVar1="+intVar1 +
                "\n intVar2="+ intVar2 +
                "\n intV3=" + intV3 +
                "\n intV4=" + intV4 +
                "\n strText1=" + strText1);

            Console.WriteLine("\n\n Press any key to terminate");
            Console.ReadLine(); // pause screen!
        }
    }
}
```

```

    }
}

```

نکاتی در مورد کد فوق:

- یک اسلش ان ، در زبانهای مشتق شده از سی به معنای new line می باشد.
- در کد فوق نحوه ی تعریف چند متغیر در یک خط و حالتی مقدار دهی مختلف را ملاحظه می کنید.
- از متد ReadLine برای نگه داشتن خروجی و مشاهده ی آن در اینجا استفاده کردیم.
- عادت کنید به صورت دندانه دار کد بنویسید. اینکار خوانایی کد را صد برابر می کند. در اینجا کدهای داخل متد main ، کاملاً چند دندانه از آکولادهای باز و بسته کردن آن جلو تر هستند.
- در کد بالا در متد WriteLine اعداد و رشته ها با هم جمع شده اند! این مورد بدلیل وجود overload های زیاد این تابع و ... میسر گشته است. اصلاً به آن دل نبندید! چون در آینده کامپایلر سی شارپ اگر چنین اعمالی را در جاهای دیگری مرتکب شوید به شدت با شما برخورد خواهد کرد!! برای جمع کردن اعداد با رشته ها حتماً باید عدد به رشته تبدیل گردد و بعد در این مورد در مقالات بعدی بحث خواهد گردید.

در مورد کلاسها و using و namespace و غیره در آینده بیشتر صحبت خواهیم کرد.

مقدمه ای بر سی شارپ : قسمت- ۳

مقدمه :

در این قسمت می خواهیم با یک سری از اصول اولیه ی شیء گرایی در سی شارپ کمی آشنا شویم. لازم به ذکر است ، بسیاری از مواردی که در این قسمت مطرح می شوند فقط برای آشنایی شما است و در آینده بیشتر بحث و مرور خواهند شد.

آشنایی با فضاها (NameSpaces) :

فضاهای نام روشی برای مدیریت کد نویسی هستند. برای مثال آنها ایجاد شده اند تا تداخلی بین نام های توابع در برنامه شما رخ ندهد. این مساله در پروژه های بزرگ خود را نشان می دهد و ممکن است دو آیتم در یک پروژه نام های یکسانی را پیدا کنند. بدین وسیله این شانس تصادم و تداخل کاهش پیدا می کند. برای ایجاد یک فضای نام به صورت زیر عمل می شود:

```

namespace anyName
{
    .....

    Class anyClassName
    {
        .....
    }

    .....
}

```

یکی از فضاها نام پایه ای در دات نت فریم ورک ، فضای نام System می باشد. برای استفاده از آن می توان از کد زیر کمک گرفت :

```
using System;
```

تمام فضاها نام به صورت پیش فرض public می باشند و در خارج از کد شما قابل دسترسی هستند. روش استفاده از آنها به صورت زیر است:

```
ProjectName.Namespace.ClassName.MemberName
```

نکته :

اگر دقت کرده باشید هنگامی که کرسر ماوس را روی هر آیتمی در منوی autocomplete نگه می دارید و یا آنرا انتخاب می کنید یک راهنمای کوچک نمایش داده می شود که در حقیقت کامنت مربوط به آن تابع می باشد. روش نوشتن چنین کامنت حرفه ای که در منوهای ویژوال استودیو ظاهر شود به صورت زیر است که بهتر است (!) قبل از هر تابع یا خاصیت یا کلاس و نوشته شود

```
///<summary>
///
///
///</summary>
```

کلاس ها :

چون سی شارپ تمام سر و کارش با کلاس ها است بنابراین باید در مورد نحوه ی تعریف و استفاده از آنها تسلط کافی داشته باشیم.

یک پروژه ی جدید console در VS.NET باز کنید و نام آنرا در ابتدا ex03 وارد نمایید. بعد از باز شدن پروژه ، از منوی Project گزینه ی Add class را انتخاب کنید تا کلاسی جدید به نام clsDate.cs را اضافه نماییم. ساختار فایل ایجاد شده توسط VS.NET به صورت زیر است :

```
using System;

namespace ex03
{
    /// <summary>
    /// Summary description for clsDate.
    /// </summary>
    public class clsDate
    {
        public clsDate()
        {
            //
            // TODO: Add constructor logic here (chashm!)
            //
        }
    }
}
```

تابع یا متد clsDate که در اینجا به صورت پیش فرض ایجاد شده است اصطلاحاً سازنده (constructor) نام دارد. این تابع هر بار که یک شیء جدید از کلاس می سازیم به صورت خودکار اجرا می شود.

از این کلاس می خواهیم برای نمایش تاریخ/ ساعت و غیره استفاده کنیم.

برای مثال می خواهیم تاریخ جاری سیستم را به صورت یک خاصیت از این کلاس دریافت کنیم. برای این منظور کد زیر را به برنامه اضافه می نماییم:

```
public string currentSystemDate
{
    get
    {
        return System.DateTime.Today.ToString() ;
    }
}
```

توضیح کد فوق :

خاصیتی را که می خواهیم از برنامه دریافت کنیم با کلمه ی کلیدی get معرفی می نماییم. هر چیزی که این قسمت برگرداند خروجی currentSystemDate خواهد بود. این دستور زبان که در بالا معرفی شد استاندارد است و در همه جا به یک صورت تعریف و بکار برده می شود. پس شکل آنرا به خاطر بسپارید. از کلمه ی کلیدی return برای برگرداندن یک خروجی از خاصیت و یا تابع استفاده می شود.

برای استفاده از این خاصیت جدید ، در فایل Class1.cs که متد main برنامه ی ما در آنجا قرار دارد به صورت زیر عمل می کنیم :

```
clsDate m_var = new clsDate(); // initialize variable
Console.WriteLine ( m_var.currentSystemDate );

Console.ReadLine(); //pause!
```

توضیح کد فوق :

برای استفاده از یک کلاس باید یک متغیر از آن را تعریف کنیم. در هر زبانی یک سری نوع های استاندارد مانند int و string و غیره وجود دارند. کلاس هم در حقیقت یک نوع داده ی بسیار بسیار قدرتمند به شمار می آید. برای تعریف یک متغیر از نوع جدید روش کار مانند سابق است. برای مثال زمانی که یک متغیر عدد صحیح را تعریف می کنید به صورت زیر عمل می شود :

```
int i=0;
```

برای تعریف یک متغیر از نوع داده ای که خودمان تعریف کرده ایم نیز باید به همین صورت عمل شود.

```
clsDate m_var = new clsDate();
```

از کلمه ی کلیدی new اینجا به صورت استاندارد برای مقدار دهی اولیه به این متغیر جدید استفاده می نماییم.

سپس به روش دستیابی به این خاصیتی که به کلاس اضافه کرده ایم می رسیم.

```
m_var.currentSystemDate
```

کلا چه یک خاصیت و یا یک متد را به کلاس اضافه نماییم برای دستیابی به آن از عملگر نقطه پس از ذکر نام متغیر تعریف شده از نوع کلاس خود ، استفاده می نماییم. برای استفاده از خاصیت ها نیازی به آوردن () بعد از ذکر نام خاصیت نمی باشد.

عموما از خاصیت ها برای برگرداندن و یا تنظیم یک مقدار ساده استفاده می شود و در آنها عملیات پیچیده ای مد نظر نمی باشد.

توضیحی در مورد System.DateTime.Today.ToString () :
استفاده از خواص :

شما به ویژگی های یک شیء با استفاده از خواص آن می توانید دسترسی پیدا کنید. یک property عضوی است که امکان دسترسی به ویژگی شیء یا کلاس را فراهم می کند. برای مثال طول یک رشته (string) ، سایز یک فونت ، عنوان یک فرم و نام یک مصرف کننده ، خاصیت هستند . بسیاری از اشیاء ذاتی دات نت فریم ورک ، خواص مفید زیادی را به همراه دارند. برای مثال شیء DateTime را در نظر بگیرید. با استفاده از خاصیت Today آن می توان تاریخ جاری سیستم را بدست آورد. برای استفاده از یک خاصیت لازم است تا کلاس تعریف کننده شیء در برنامه مهیا باشد. منظور همان استفاده از فضای نام مربوطه می باشد. پس از وارد کردن فضای نام کلاس مورد نظر می توانید از شیء و خواص آن استفاده کنید. دو راه وجود دارد یا به صورت کامل تمام موارد باید ذکر شوند مانند System.DateTime.Now و یا با وارد کردن فضای نام System کوتاه سازی صورت می گیرد. برای استفاده از هر متد و یا شیء ایی در سی شارپ باید این شیء قابل دسترسی باشد. برای مثال شیء Console که از آن برای چاپ کردن خروجی بر روی صفحه ی نمایش استفاده می کنیم در فضای نام System واقع شده است. یا باید در ابتدای برنامه ذکر کرد ; using System و سپس خیلی راحت از این شیء استفاده کرد و یا می توان اینکار را انجام نداد و نوشت : System.Console و الی آخر. با ذکر فضای نام در ابتدا با استفاده از using می توان خلاصه نویسی کرد.

نتیجه ی نهایی مثال این فصل :

محتویات فایل Class1.cs :

```
using System;

namespace ex03
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            clsDate m_var = new clsDate(); // initialize variable
            Console.WriteLine ( m_var.currentSystemDate );

            Console.ReadLine(); //pause!
        }
    }
}
```

محتویات فایل clsDate.cs که به برنامه اضافه کردیم:

```
using System;

namespace ex03
{
    /// <summary>
```



```

/// Summary description for clsDate.
/// </summary>
public class clsDate
{
    public clsDate()
    {
        //
        // TODO: Add constructor logic here
        //
    }

    public string currentSystemDate
    {
        get
        {
            return System.DateTime.Today.ToString() ;
        }
    }
}

```

مقدمه ای بر سی شارپ : قسمت-۴

ساختارهای تصمیم گیری :

در بسیاری از موارد هنگام برنامه نویسی لازم است تا از عبارات شرطی استفاده کنیم. برای انجام اینکار دو روش عمده وجود دارد. استفاده از if و یا switch. از if بیشتر برای مقایسه هایی تکی و کوچک استفاده می شود و حاصل مقایسه ی آن یا true است و یا false. از عبارت switch هنگامی استفاده می شود که مقایسه های متعددی باید در مورد یک مقدار صورت گیرد.

هر دو عبارت if و switch توسط عبارتهایی Boolean کنترل می شوند (true و یا false). در هنگام استفاده از if اگر عبارت Boolean حاصل اش true باشد اولین قسمت شرط اجرا می شود و سپس برنامه از انتهای if ادامه پیدا می کند. اگر حاصل عبارت Boolean مساوی false باشد کنترل برنامه به قسمت else منتقل می شود.

مثال :

یک پروژه ی جدید console باز کنید و نام آنرا ex04 بگذارید. سپس کد زیر را در آن وارد و اجرا کنید :

```

using System;

namespace ex04
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            Console.WriteLine("Enter 1 character to be evaluated");

            char cUserInput = (char) Console.Read();

```

```

        if ( char.IsDigit( cUserInput ) )
            Console.WriteLine("The char is a number!");
        else
            Console.WriteLine("The char is not a number!");
    }
}

```

نکاتی در مورد کد فوق :

- ۱- سی شارپ به کوچکی و بزرگی حروف حساس است . برای مثال cUserInput با cUserinput فرق می کند.
- ۲- حتما باید بعد از if پرانتزها ذکر گردد.
- ۳- حتما باید داخل if یک عبارت Boolean ذکر شود مانند if(x>5) .
- ۴- در سی شارپ مقایسه ی تساوی دو عبارت با == و انتساب با = انجام می شود. (موارد ۱ و ۴ مواردی هستند که اغلب تازه کاران با آن مشکل دارند!) برای مثال if(i==3) صحیح است اما if(i=3) در سی شارپ معنایی ندارد.
- ۵- اگر بعد از if یک خط کد قرار گیرد نیازی به آوردن آکولاد ها نیست. هنگامی نیاز به آکولادها می باشد که بیش از یک خط باید بعد از if قرار گیرد.
- ۶- در سی شارپ همانند اسلاف خودش برای تبدیل نوع های داده ای می توان به صورت زیر نیز عمل کرد : ; Console.Read() (char) یعنی دریافتی Read به char تبدیل می شود . در این مورد باز هم صحبت خواهد شد.
- ۷- همانطور که ذکر شد در سی شارپ همه چیز شیء است حتی نوع های پایه ای مانند char . با استفاده از متد IsDigit آن می توان چک کرد که آیا ورودی آن عدد است یا خیر؟ (در مورد متدها صحبت خواهد شد)

استفاده از switch :

بهتر است این مورد را با یک مثال دنبال کنیم. پروژه ی سی شارپ جدیدی به نام ex05 در حالت console در VS.NET باز کنید. در اینجا می خواهیم یک کلاس جدید تعریف کرده و توسط خاصیتی که در آن ایجاد می کنیم متوجه شویم روز جاری مطابق سیستم چه روزی است . یک کلاس جدید از منوی پروژه ، با استفاده از گزینه ی Add class به برنامه اضافه کنید و نام آنرا در ابتدا clsDate بگذارید.

```

using System;

namespace ex05
{
    /// <summary>
    /// Summary description for clsDate.
    /// </summary>
    public class clsDate
    {
        public clsDate()
        {
            //
            // TODO: Add constructor logic here
            //
        }

        public string systemDayOfWeek
        {
            get

```

```

{
    string res="";
    switch( System.DateTime.Now.DayOfWeek.ToString())
    {
        case "Saturday" :
            res = "شنبه";
            break;

        case "Sunday" :
            res = "شنبه يك" ;
            break;

        case "Monday":
            res = "دوشنبه";
            break;

        case "Tuesday":
            res = "شنبه سه";
            break;

        case "Wednesday":
            res = "شنبه چهار";
            break;

        case "Thursday":
            res = "شنبه پنج";
            break;

        case "Friday":
            res = "جمعه" ;
            break;

    }

    return res ;
}
}
}

```

هنگام ذخیره کردن این کد ویژوال استودیو به شما اخطار می دهد که کد دارای حروف یونیکد است. از منوی فایل گزینه ی advanced save options را انتخاب کنید. در اینجا می توان نوع ذخیره سازی را یونیکد انتخاب کرد.

برای استفاده از کلاس فوق مانند مطالبی که در قسمت قبل گفته شد عمل می کنیم :

```

using System;

namespace ex05
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            clsDate m_var = new clsDate();

```

```

        Console.WriteLine( m_var.systemDayOfWeek );
        Console.ReadLine();
    }
}

```

هر چند حالت console یونیکد را پشتیبانی نمی کند ولی اصل برنامه برای ما مهم است و در آینده بیشتر از آن استفاده خواهیم کرد.

همانطور که ملاحظه کردید اگر از switch استفاده نمی شد باید از ۷ عدد if استفاده می گردید که اصلاً ظاهر حرفه ای و شکیلی نداشت! با استفاده از عبارت زیر کار مقایسه شروع می شود. روز سیستم در یافت شده و وارد بدنه ی switch می گردد. سپس توسط case ها چک می شود تا تساوی آن با عبارت بعد از case به اثبات برسد.

```
switch ( System.DateTime.Now.DayOfWeek.ToString() )
```

اگر هر کدام از عبارات بعد از case صحیح بودند کار پس از آن که در اینجا انتساب است انجام شده و سپس توسط break کنترل برنامه از switch خارج می شود و ادامه ی کار دنبال می گردد. اگر هیچکدام از case ها صحیح نبودند می توان از گزینه ی default هم در صورت نیاز استفاده کرد. این حالت در یک چنین مواقعی اجرا می گردد.

مقدمه ای بر سی شارپ : قسمت- ۵

آرایه ها در سی شارپ :

هنگامی آرایه ها ایجاد می شوند که بخواهیم با مجموعه ای از اطلاعات همجنس کار کنیم. برای نمونه از یک آرایه برای ذخیره تعدادی کاراکتر می خواهیم استفاده نماییم. آرایه ها هم یک نوع متغیر هستند پس باید تعریف و مقدار دهی اولیه شوند ، نوع و تعداد اعضای آنها نیز باید معین گردد. فرض کنید ۱۰ داده ی هم جنس داریم (برای مثال رشته (string)) و می خواهیم آنها را ذخیره کنیم. یا می توان ۱۰ متغیر مختلف را تعریف کرد و سپس تک تک آنها را مقدار دهی نمود و یا یک آرایه تعریف نمود و سپس در خانه های مختلف آن این ده عضو را چید. این مطلب زمانی حائز اهمیت می شود که داده های همجنس و به نوعی مرتبط ما تعداد زیادی داشته باشند.

برای تعریف آرایه چندین راه مختلف وجود دارد :

برای تعریف آرایه ابتدا نوع آنرا مشخص می کنید سپس [] را باید جلوی تعریف نوع بگذارید این دستور زبان است و چون چرا ندارد! در زبان سی کمی متفاوت بود. این کروش ها بعد از نام متغیر می آمدند. و سپس در اینجا نام یک متغیر را که بعداً به آن ارجا می دهیم خواهیم گذاشت. برای مثال

```
int[] table; // not int table[];
```

حد پایین آرایه صفر بوده برای مثال اگر آرایه chrData[] ده عضو داشته باشد، اولین عضو آن chrData[0] و آخرین عضو آن chrData[9] است.

مطلب دیگری که در مورد آرایه ها خیلی مهم است اندازه ی آن است. یعنی یک آرایه حاوی چند خانه ی خالی است که ما اجازه داریم آنرا پر کنیم. مثال :

```
int[] numbers; // declare numbers as an int array of any size
numbers = new int[10]; // numbers is a 10-element array
numbers = new int[20]; // now it's a 20-element array
```

۱- تعریف آرایه ای از رشته ها و مقدار دهی اولیه آن.

```
String[] strData = new string[2];
```

۲- تعریف و مقدار دهی اولیه

```
string [] strData = { "1234","abcd" };
```

که آرایه ای از نوع رشته ای به طول ۲ عضو با مقدار دهی اولیه ایجاد شده است. در این حالت نیازی به تعیین طول آن نمی باشد.

۳- روشی دیگر برای مقدار دهی اولیه

```
strData[0] = "1234";
strData[1] = "abcd";
```

مثال : یک پروژه ی جدید Console سی شارپ را باز کنید و نام آنرا در ابتدا ex06 بگذارید. در این مثال می خواهیم نحوه ی کار با آرایه ها را مرور کنیم :

```
using System;

namespace ex06
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            string[] sGoalList = new string[3];
            string sReplyStatement = "You have choosen Goal ";

            // Store goals in the array
            sGoalList[0] = "Hike the Appalachian Trail";
            sGoalList[1] = "Run the marathon";
            sGoalList[2] = "Give $1 million to worthwhile causes";

            // Store response to goals in the array
            //(declaring and initializing on same line)
            string[] sGoalResponse = {
                "If you are staring from GA, you should get  "
                + "started in early spring, so you will "+
                "not get caught in snow.",
                "Make sure that you have a good pair of shoes.",
                "Start saving as soon as possible."};

            // Give the user a list of goals to choose from
            Console.WriteLine("GOAL LIST");

            for(int i = 0; i < sGoalList.Length; i++)
            {
                Console.WriteLine("Goal " + i +
                                   " - " + sGoalList[i]);
            }
        }
    }
}
```

```
// Request the user to choose a goal.
Console.WriteLine ( ""); // Write an empty line for space
Console.Write("Please choose the number of the "
    + "goal that you want to achieve [0,1,2]: ");

Console.ReadLine();

    }
}
}
```

نکاتی در مورد کد فوق :

- ۱- نحوه ی استفاده از عملگر + را برای اتصال رشته های بلند در کد فوق می توان دید.
- ۲- در سی شارپ پایان خط سمی کولون می باشد. بنابراین نگرانی در مورد چند خطی شدن یک دستور وجود ندارد.
- ۳- هنگامی که آرایه ای را با مقادیر درون آکولادها ، مقدار دهی اولیه می کنید لزومی ندارد طول آن آرایه را مشخص کنید ؛ مانند آرایه sGoalResponse در بالا. در غیر اینصورت حتما باید طول یک آرایه را که معرف تعداد خانه های خالی آن است ، معرفی کنید مانند آرایه sGoalList .
- ۴- فعلا حلقه ی for را در این مثال بخاطر داشته باشید تا در مقاله ی بعدی راجع به آن صحبت کنیم.

مقدمه ای بر سی شارپ : قسمت- ۶

حلقه ها در سی شارپ :

مقدمه :

اگر نیاز باشد تا قطعه ای از کد بیش از یکبار اجرا شود نیاز به استفاده از حلقه ها می باشد. برای مثال فرض کنید آرایه ای به طول ۱۰۰۰ تعریف کرده اید. اکنون می خواهید آنرا با هزار عدد متوالی پر کنید. بدیهی است که روش زیر کارآمد نیست! :

```
int[] intData = new int[1000];
intData[0]=0;
.
.
.
intData[999]=1000;
```

نوشتن این خطوط متوالی احتمالا با کپی و پیست و اصلاح آن حداقل نیم ساعت طول می کشد! بنابراین نیاز به وسیله ای حس می شود که بتوان بوسیله ی آن امثال اینگونه کارها را انجام داد.

تعریف حلقه ها و استفاده از آنها :

برای تعریف حلقه ها ابزارهای متعددی مانند for , do , while وجود دارند. استفاده و انتخاب آنها بستگی به سلیقه ی شما و منطق برنامه دارد. در هر حال یک مساله بدیهی است که همواره بیش از یک راه حل برای یک مساله وجود خواهد داشت.

استفاده از حلقه ی for :

عموما کدنویسی را با کد نویسی می توان آموخت! بنابراین در مورد انواع حلقه ها مثالهایی ارائه خواهد گردید.
یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex07 انتخاب نمایید. سپس کد زیر را درون آن بنویسید :

```
using System;

namespace ex07
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            int[] intData = new int[1000];

            for (int i=0 ; i<1000 ; i++ )
                intData[i]=i;

            for(int i=0 ; i< intData.Length ; i++)
            {
                int j = intData[i];
                Console.WriteLine("intData[" + i + "]= " + j);
            }

            Console.ReadLine();
        }
    }
}
```

توضیحاتی در مورد کد فوق :

- ۱- برای تعریف حلقه ی for همانطور که می بینید باید تعداد بار اجرای حلقه (اینجا از 0 تا 999 است) و همچنین نحوه ی رسیدن از 0 به 1000 را مشخص کرد (در اینجا ++i است یعنی هر بار یک واحد به شمارشگر حلقه اضافه می شود.)
- ۲- در زبان سی ++i یعنی i=i+1 و --i یعنی i=i-1 و کلا i=n یعنی i=i-n و به همین ترتیب. برای مثال i*=n یعنی i=i*n و i+=n یعنی i=i+n و ...
- ۳- اگر پس از حلقه ی for یک خط کد داشته باشیم نیازی به آکولاد نیست (مانند قسمت اول کد). ولی اگر تعداد خطوط مربوط به بدنه ی for زیاد بود باید حتماً از آکولاد استفاده شود (مانند قسمت دوم کد). (این قاعده ای کلی است در زبانهای مشتق شده از زبان سی در مورد هر چیزی!)
- ۴- فرض کنید در قسمت اول کد بالا بجای ۱۰۰۰ می نوشتید ۱۰۰۱. سریعاً با یک خطای زمان اجرا مواجه می شدید. زیرا می خواستید به عضوی از آرایه دسترسی پیدا کنید که تعریف نشده است. راه مدرن چک کردن این مسائل استفاده از خاصیت Length آرایه است که در قسمت دوم کد در عمل مشاهده می نمایید. همیشه از این روش استفاده کنید.
- ۵- حلقه ی اول یعنی اینکه کار پر کردن آرایه intData را از صفر تا 999 یکی یکی (i++) انجام بده.

استفاده از حلقه ی while :

يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا در ابتدا ex08 انتخاب نماييد. سپس كد زير را درون آن بنويسيد :

```
using System;

namespace ex08
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            int n = 1;

            while (n < 6)
            {
                Console.WriteLine("Current value of n is {0}", n);
                n++;
            }

            Console.ReadLine();
        }
    }
}
```

توضيحاتي در مورد كد فوق :

- ۱- حلقه ي while در بالا كار انجام حلقه را تا هنگامي انجام مي دهد كه شرط ذكر شده در ابتداي آن صادق و برقرار باشد. يعني در حلقه ي فوق تا وقتي $n < 6$ است اين حلقه ادامه خواهد يافت.
- ۲- حلقه ي while صفر يا بيشتر بار ممكن است اجرا شود.
- ۳- در كد فوق از {0} استفاده گرديده است. متد WriteLine به شما اين اجازه را مي دهد كه n تا آرگومان براي آن تعريف كنيد و مقادير هر کدام را كه خواستيد در كد نمايش دهيد از {X} استفاده كنيد. در اين مورد مقدار آرگومان x ام نمايش داده مي شود.

استفاده از حلقه ي do :

يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا در ابتدا ex09 انتخاب نماييد. سپس كد زير را درون آن بنويسيد :

```
using System;

namespace ex09
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            int x;
            int y = 0;

            do
            {
                x = y++;
            }
        }
    }
}
```



```

        Console.WriteLine(x);

        }while(y < 5);

        Console.ReadLine();

    }

}

```

توضیحاتی در مورد کد فوق :

- ۱- این حلقه به حلقه ی do...while معروف است و هر دو جزء آن باید ذکر گردد.
- ۲- این حلقه تا زمانی که شرط ذکر شده در قسمت while صحیح است ادامه می یابد.
- ۳- این حلقه در ابتدای کار بدون توجه به قسمت while حداقل یکبار اجرا می شود. (مثال زیر را اجرا نمایید)

```

int n = 10;
do
{
    Console.WriteLine("Current value of n is {0}", n);
    n++;
} while (n < 6);

```

استفاده از حلقه ی foreach :

یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex10 انتخاب نمایید. سپس کد زیر را درون آن بنویسید :

```

using System;

namespace ex10
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            int odd = 0, even = 0;
            int[] arr = new int [] {0,1,2,5,7,8,11};

            foreach (int i in arr)
            {
                if (i%2 == 0)
                    even++;
                else
                    odd++;
            }

            Console.WriteLine(
                "Found {0} Odd Numbers, and {1} Even Numbers.",
                odd, even) ;

            Console.ReadLine();

        }

    }
}

```

}

توضیحاتی در مورد کد فوق :

- ۱- از foreach برای حرکت در بین اعضای یک آرایه (مانند مثال بالا) و یا مجموعه ایی از اشیاء استفاده می شود (روشی شکیل ، مدرن و مطمئن! و تقریباً به ارث رسیده از ویژوال بیسیک!!).
- ۲- در زبانهای مشتق شده از C ، عملگر % ، باقیمانده را محاسبه می کند.
- ۳- در کد فوق با استفاده از حلقه ی foreach تک تک اعضای آرایه در مورد زوج و یا فرد بودند مورد بررسی قرار گرفته اند و تعداد اعضای زوج و فرد در آخر نمایش داده می شود.

مقدمه ای بر سی شارپ : قسمت- V

دو مورد تکمیلی در مورد حلقه ها در سی شارپ :

- ۱- هر جایی خواستید به هر دلیلی حلقه را پایان دهید می توانید از دستور break استفاده کنید. در این حالت به صورت آنی حلقه خاتمه یافته و کدهای ادامه ی برنامه پس از حلقه اجرا می شوند.
- ۲- نحوه ی استفاده از دستور continue : فرض کنید حلقه ی شما در راند ۱۵ خودش است! حالا در این راند شما می خواهید یک سری از دستورات درون حلقه اجرا نشوند و حلقه به راند بعدی منتقل شده و کارش را ادامه دهد. اینجا است که از دستور continue استفاده می شود. بهتر است به یک مثال ساده در این زمینه توجه کنیم.

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex11 انتخاب نمایید. سپس کد زیر را درون آن بنویسید :

```
using System;

namespace ex11
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            Console.WriteLine(
                "for (int i = 1; i <= 100; i++) -> break at i==5" );
            for (int i = 1; i <= 100; i++)
            {
                if (i == 5)
                    break;
                Console.WriteLine(i);
            }
            Console.ReadLine();

            Console.WriteLine(
                "for (int i = 1; i <= 10; i++) -> continue if i<9" );
            for (int i = 1; i <= 10; i++)
            {
                if (i < 9)
                    continue;
                Console.WriteLine(i);
            }
        }
    }
}
```

```

        Console.ReadLine();
    }
}

```

موارد تکمیلی مربوط به رد و بدل کردن مقادیر به/از کلاس ها :

در قسمت بعدی می خواهیم خاصیتی را تعریف کنیم که یک مقدار را از کاربر می گیرد و در برنامه می توان توسط قسمت های دیگر از آن استفاده کرد.

ابتدا یک متغیر عمومی باید در سطح کلاس تعریف کرد تا مقدار دریافت شده توسط set را در خود نگاه داری کند (در مورد scope متغیرها (متغیرهای عمومی و محلی و امثال اینها) در هنگام معرفی توابع بیشتر بحث خواهد شد). سپس از طریق کلمه ی کلیدی value مقدار دریافت شده به متغیر انتساب می یابد و چون در سطح کلاس عمومی است در تمام کلاس قابل دسترسی است.

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex12 انتخاب نمایید. سپس از منوی پروژه یک کلاس جدید به آن اضافه نمایید (به نام clsDate) و کد زیر را درون آن بنویسید :

```

using System;

namespace ex12
{
    public class clsDate
    {
        private int Year;

        public clsDate()
        {
        }

        public int setYear
        {
            set
            {
                Year = value;
            }
        }

        public bool IsLeapYear
        {
            get
            {
                return System.DateTime.IsLeapYear(Year);
            }
        }
    }
}

```

برای استفاده از آن در متد main برنامه به صورت زیر عمل می کنیم:

```
using System;

namespace ex12
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            clsDate m_var = new clsDate();

            m_var.setYear = 1990;

            if (m_var.IsLeapYear)
                Console.WriteLine("1990 is a leap year.");
            else
                Console.WriteLine("1990 is not a leap year.");

            Console.ReadLine();
        }
    }
}
```

توضيحاتي در مورد كد فوق:

- ۱- نحوه ي تعريف متغير از يك كلاس جزو اساسي ترين قسمت هاي كار با يك كلاس محسوب مي شود كه در قسمت هاي پيشين نيز معرفي گرديد.
- ۲- هنگامي كه از if استفاده مي كنيم لزومي ندارد حتما بنويسيم `m_var.IsLeapYear==true` . همين كه اين خاصيت ذكر مي شود در وهله ي اول true بودن آن چك خواهد شد.
- ۳- نحوه ي مقدار دهني به يك خاصيت را هم در كد فوق ملاحظه مي نماييد. در هنگام استفاده از خاصيت ها نيازي به آوردن پرانتزها () در مقابل نام آنها وجود ندارد.
- ۴- براي مرور ، نحوه ي معرفي خاصيت ها با get نيز بيان گرديد. با استفاده از set و get مي توان به كلاس ها ، مقادير متغيرها را پاس كرد و يا مقداري را دريافت نمود.

مقدمه اي بر سي شارپ : قسمت- ۸

تعريف متدها در سي شارپ

در اين قسمت به يكي از مهمترين مباحث برنامه نويسي سي شارپ مي رسيم. متدها در سي شارپ و يا همان توابع در زبان C ، اعزاي يك شيء يا كلاس هستند و مجموعه اي از يك سري از كارها را انجام مي دهند. فرض كنيد در برنامه ي شما ، قسمتي بايد يك عمليات رياضي خاص را انجام دهد و اين قسمت از كد كه شامل چندين خط نيز مي گردد بايد بارها و بارها در برنامه صدا زده شود. براي نظم بخشيدن به برنامه ، آنها را مي توان به صورت توابع بسته بندي كرد و بجاي نوشتن چندين خط تكراري، فقط نام اين بسته (تابع) و پارامترهاي آن را فراخواني نمود.

در سي شارپ يك تابع به صورت زير تعريف مي شود :

```
(نوع و اسامي پارامترها) نام تابع    نوع خروجي تابع    سطح دسترسي به تابع
{
    بدنه ي تابع
}
```

برای تعریف یک متد یا تابع ابتدا سطح دسترسی به آن مانند `public` و `private` سپس نوع خروجی تابع مانند `void` (هیچی) ذکر می گردد که داخل این پرانتزها می توان ورودی های تابع یا بقولی آرگومان های ورودی را معرفی کرد. سپس تابع باید با `{` شروع و با `}` خاتمه یابد.

برای مثال :

```
public int myFunc( int x )
{
.....
}
```

هر تابعی می تواند صفر تا تعداد بیشمار آرگومان ورودی و صفر تا تعداد بیشمار خروجی داشته باشد. بوسیله یک تابع می توان پیچیدگی کار را مخفی کرد و صرفا با صدا زدن نام آن ، یک سری از عملیات را انجام داد. گاهی از اوقات لازم می شود دو یا چند تابع با یک نام داشته باشیم بطوریکه پارامترهای ورودی یا مقادیر خروجی و یا نوع آرگومان های ورودی آنها با هم متفاوت باشد به این کار `overloading` می گویند. بسیاری از کلاس های دات نت فریم ورک متدها و یا توابع مفید حاضر و آماده ای را دارند. برای مثال کلاس `DateTime` ، متدی به نام `ToLongDateString` دارد که تاریخ را به صورت یک رشته طولانی بر می گرداند.

توابع `void` :

توابعی که با نوع `void` معرفی می شوند هیچ خروجی ندارند و در زبان ویژوال بیسیک به آنها `sub` و در دلفی به آنها `procedure` می گویند.

بازگرداندن یک مقدار از یک تابع :

پس از اینکه عملیات یک مجموعه از کدها درون تابع به پایان رسید با استفاده از کلمه `return` می توان خروجی تابع را معرفی کرد. لازم به ذکر است ، هرجایی این کلمه `return` ذکر شود کار تابع خاتمه می یابد.

بهتر است موارد فوق را با چند مثال مرور کنیم :

مثال : یک برنامه ی سی شارپ جدید `console` را در `VS.NET` باز کنید و نام آنرا در ابتدا `ex13` انتخاب نمایید. در اینجا می خواهیم تابعی را تعریف کنیم که سه برابر جذر یک عدد را بر می گرداند.

```
using System;

namespace ex13
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            Console.WriteLine( int3SQL(3) );
            Console.ReadLine();
        }

        public static double int3SQL( double intInput )
        {
            double i=0;
            i = Math.Sqrt( intInput );
            return i;
        }
    }
}
```

```

    }
}

```

توضیحاتی در مورد کد فوق :

- ۱- از شیء Math در سی شارپ می توان برای انجام یک سری عملیات ریاضی ابتدایی استفاده کرد. در اینجا از متد جذر گرفتن آن استفاده شده است.
- ۲- در تعریف تابع خودمان از کلمه ی کلیدی static استفاده شده است. درون تابع Main نمی توان توابع غیر استاتیک را فراخوانی کرد. فعلا این نکته را بخاطر را داشته باشید تا در مقالات بعدی بیشتر راجع به آن صحبت شود.
- ۳- بد نیست تابع تعریف شده را کمی بیشتر آنالیز کنیم :

```

public static double int3SQL( double intInput )
{
    double i=0;
    i = Math.Sqrt( intInput );
    return i;
}

```

ابتدا سطح دسترسی به تابع ذکر شده است. پابلیک ، یعنی این تابع خارج از کلاس یک برنامه نیز قابل دسترسی است. سپس از کلمه ی static استفاده گردیده که توضیح مختصری را در مورد آن ملاحظه کردید. در ادامه نوع خروجی تابع که در اینجا double می باشد معرفی گردیده است. دقت کنید که حتما باید نوع تعریف شده با مقداری که یک تابع بر می گرداند یکسان باشد و گرنه با یک خطا برنامه متوقف می شود. سپس نام تابع تعریف شده است. داخل پرانتزها نوع و نام آرگومانهای ارائه شده است که در بدنه ی تابع استفاده می گردد. اگر به تعداد بیشتری پارامتر و یا آرگومان نیاز بود می توان آنها را با ، از هم جدا کرد. پس از اینکه عملیات تابع خاتمه می یابد با استفاده از return این خروجی را معرفی می نماییم. برای استفاده از این تابع به سادگی نام تابع و سپس پرانتزها به همراه یک عدد دلخواه را می نویسم که آنرا در متد Main برنامه می توان مشاهده کرد.

تعریف توابع در کلاس های دیگر برنامه و نحوه ی استفاده از آنها :

یکی از زیبایی های برنامه نویسی شیء گرا نظم و ترتیب و بسته بندی کارها می باشد که اصطلاحا در اینجا به آن encapsulation می گویند. یعنی ما یک سری از توابع و خواص را درون کپسولی به نام کلاس قرار می دهیم تا به سادگی بارها و بارها از آن استفاده نماییم. برای اینکار به سادگی یک توابع را به صورت معمول درون کلاس تعریف می نماییم و سپس همانند خواص که در مورد آنها صحبت شد ، از توابع می توان استفاده کرد با این تفاوت که هنگام کار با توابع حتی اگر آنها هیچ آرگومان و یا پارامتر ورودی هم نداشته باشند ذکر پرانتزها الزامی است.

مثالی دیگر در این زمینه :

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex14 انتخاب نمایید. سپس از منوی پروژه یک کلاس جدید را به برنامه اضافه نمایید (نام آنرا clsTools بگذارید) .

```

using System;

namespace ex14
{
    public class clsTools
    {
        public clsTools()
        {
        }
    }
}

```

```

public uint intCalc ( uint a , uint b )
{
    uint    c = Math.Min (a,b);
    double  x = Math.Sqrt(c) ;
    uint    w = Convert.ToUInt32 ( x);
    return  w;
}
}

```

سپس در متد Main برنامه می توان به صورت زیر از آن استفاده کرد :

```

using System;

namespace ex14
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            clsTools m_var = new clsTools();
            Console.WriteLine( m_var.intCalc(4,9));
            Console.ReadLine();
        }
    }
}

```

توضیحاتی در مورد کد فوق :

- ۱- تابع intCalc ما دو عدد صحیح مثبت را می گیرد و سپس جذر کوچکترین دو عدد ورودی را محاسبه می کند.
- ۲- برای تبدیل نوع های عددی مختلف به هم می توان از شیء Convert استفاده کرد.
- ۳- بدون استفاده از شیء Convert یکبار برنامه را اجرا کنید و دلیل خطای بوجود آمده را بیان نمایید.

بحث توابع ادامه دارد...

مقدمه ای بر سی شارپ : قسمت- ۹

چگونه از یک تابع بیش از یک خروجی دریافت کنیم.

ظاهراً به نظر می رسد که توابع فقط می توانند یک return داشته باشند و بلافاصله پس از فراخوانی return کار تابع پایان یافته است. در سی شارپ دو کلمه ی کلیدی به نام های ref و out اضافه شده اند که این امر را ساده تر می کنند.

استفاده از کلمه ی کلیدی out :

از out در تعریف تابع قبل از معرفی نوع آرگومان ورودی استفاده می کنیم. در این حالت بجای اینکه به این آرگومان ، آرگومان ورودی بگوییم ، می توان آنرا آرگومان خروجی نامید. تا یک مثال را در این زمینه با هم مرور نکنیم این مورد مفهوم نخواهد بود :

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex15 انتخاب نمایید. سپس کد زیر را درون آن بنویسید :

```

using System;

namespace ex15
{
    class Class1
    {
        public static int TestOut(out char i)
        {
            i = 'b';
            return -1;
        }

        [STAThread]
        static void Main(string[] args)
        {
            char i;    // variable need not be initialized
            Console.WriteLine(TestOut(out i));
            Console.WriteLine(i);
            Console.ReadLine();
        }
    }
}

```

توضیحاتي در مورد کد فوق :

- ۱- در تابع TestOut آرگومان i از با کلمه ي کلیدی out مشخص شده است. يعني اینکه درون تابع هر گونه تغییری روی i انجام شود ، خارج از تابع قابل دسترسی است.
- ۲- توابعی که دارای آرگومانهایی تعریف شده با کلمه ي کلیدی out هستند نیز می توانند از return هم استفاده کنند. همانند مثال فوق.

استفاده از کلمه ي کلیدی ref :

این کلمه ي کلیدی نیز دقیقا همانند out عمل می کند و نحوه ي تعریف و استفاده از آن نیز مشابه است با این تفاوت که آرگومانی که به این نوع توابع فرستاده می شود باید مقدار دهی اولیه شده باشد.

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز کنید و نام آنرا در ابتدا ex16 انتخاب نمایید. سپس کد زیر را درون آن بنویسید :

```

using System;

namespace ex16
{
    class Class1
    {
        public static void FillArray(ref int[] arr)
        {
            // Create the array on demand:
            if (arr == null)
                arr = new int[10];
            // Otherwise fill the array:
            arr[0] = 123;
        }
    }
}

```



```

        arr[4] = 1024;
    }

    [STAThread]
    static void Main(string[] args)
    {
        // Initialize the array:
        int[] myArray = {1,2,3,4,5};

        // Pass the array using ref:
        FillArray(ref myArray);

        // Display the updated array:
        Console.WriteLine("Array elements are:");
        for (int i = 0; i < myArray.Length; i++)
            Console.WriteLine(myArray[i]);

        Console.ReadLine();
    }
}

```

توضیحاتی در مورد کد فوق :

- ۱- همانطور که ملاحظه می کنید در اینجا هنگام استفاده از تابع FillArray باید آرگومانی را که می خواهیم به آن پاس کنیم مقدار دهی اولیه کنیم.
- ۲- پس می توان نتیجه گرفت آرگومانهایی که با out تعریف می شوند به صورت خالص خروجی هستند و نیازی به مقدار دهی اولیه هنگام استفاده از آنها وجود ندارد. از ref هنگامی استفاده می کنیم که بخواهیم روی متغیر موجود و مقدار دهی شده ی خارج از تابع ، درون تابع عملیاتی صورت گیرد و سپس همان متغیر دستکاری شده ، عودت داده شود.

تعریف تابعی با تعداد آرگومانهای نامعلوم :

گاهی از اوقات نیاز است تا تابعی تعریف کنیم که تعداد آرگومانهای آن متغیر باشند. برای این منظور از کلمه ی کلیدی **params** استفاده می شود. دونکته در اینجا حائز اهمیت است:

- ۱- در هر تابعی تنها می توان یکبار از **params** استفاده کرد.
- ۲- پس از بکار بردن **params** دیگر نمی توان هیچ آرگومانی را تعریف کرد.

یکی از مثالهایی که در این زمینه می توان ارائه داد استفاده از آرایه ها به عنوان آرگومان ورودی است. در این حالت یا می توان یک آرایه را به صورت کامل به تابع معرفی کرد و یا تنها نام آنرا به تابع پاس کرد. مثال زیر را ملاحظه کنید :

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex17 انتخاب نمایید. سپس کد زیر را درون آن بنویسید :

```

using System;

namespace ex17
{
    class Class1
    {

```

```

public static void UseParams(params int[] list)
{
    for ( int i = 0 ; i < list.Length ; i++ )
        Console.WriteLine(list[i]);
    Console.WriteLine();
}

[STAThread]
static void Main(string[] args)
{
    UseParams(1, 2, 3);

    int[] myarray = new int[3] {10,11,12};
    UseParams(myarray);

    Console.ReadLine();
}
}

```

توضیحاتی در مورد کد فوق :

- ۱- در تابع main به دو صورت از تابع UseParams ما استفاده شده است. یا اینکه خیلی ساده هر تعداد آرگومان را می توان به تابع فرستاد و یا اینکه در ادامه آرایه ایی رسماً تعریف و سپس به تابع فرستاده شود.
- ۲- نحوه ی تعریف و استفاده از آرایه ها به صورت آرگومان ورودی را نیز می توان در مثال فوق آموخت.

مقدمه ای بر سی شارپ : قسمت- ۱۰

مبحث overloading :

گاهی از اوقات لازم است تا نگارش های مختلفی از یک تابع داشته باشیم. برای مثال تعریف سه تابع با یک نام اما با آرگومانهای مختلف. به این نوع توابع و یا متدها اصطلاحاً Overloaded Methods می گویند. (فکر کنم آنرا سربارگذاری توابع ترجمه کرده اند!) برای مثال :

```

void myMethod(int p1);
void myMethod(int p1, int p2);
void myMethod(int p1, string s1);

```

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex18 انتخاب نمایید. سپس کد زیر را درون آن بنویسید :

```

using System;

namespace ex18
{
    class Class1

```

```

{
    [STAThread]
    static void Main(string[] args)
    {
        writeIT();

        writeIT(12);

        Console.ReadLine();
    }

    public static void writeIT()
    {
        Console.WriteLine(" writeIT() Ver." );
    }

    public static void writeIT(int intI)
    {
        Console.WriteLine(" writeIT(intI) Ver. = " + intI );
    }
}
}

```

توضیحاتي در مورد كد فوق :

- ۱- نحوه ي تعريف دو تابع با يك نام را ملاحظه مي نماييد. اينكار در زبان سي ممنوع است!
- ۲- كامپايلر به صورت هوشمند بر اساس نوع و تعداد آرگومانهاي ورودی ، ورژن مناسب را انتخاب و اجرا مي كند.

نمونه ي ضعيفي از اين بحث در وي بي ۶ به صورت تعريف توابعي با پارامترهاي Optional وجود داشت .

مباحث تکميلي آرايه ها (آرايه هاي چند بعدي):

آرايه هاي معمولي (يك بعدي) را مي توان يك ردیف با تعدادي خانه خالي آماده ي پر شدن در نظر گرفت. آرايه ي دوبعدي را مي توان مانند يك جدول تشکيل شده از ردیف ها و ستون ها در نظر گرفت و الي آخر...
سي شارپ دو نوع آرايه ي چند بعدي را پشتياني مي كند : rectangular and jagged
در يك آرايه ي rectangular هر ردیف ، طولش با ردیف بعدي يکي است. آرايه ي jagged در حقيقت آرايه ايي از آرايه ها است ، بنابرین هر کدام از آنها مي تواند طول مختلفي داشته باشد.

تعريف يك آرايه ي دوبعدي به صورت زیر است :

```
type [,] array-name
```

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا در ابتدا ex19 انتخاب نماييد. سپس كد زیر را درون آن بنويسيد :

```

using System;

namespace ex19
{

```

```

class Class1
{
    [STAThread]
    static void Main(string[] args)
    {
        const int rows = 4;
        const int columns = 3;
        // declare a 4x3 integer array
        int[,] rectangularArray = new int[rows, columns];
        // populate the array
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < columns; j++)
            {
                rectangularArray[i, j] = i + j;
            }
        }
        // report the contents of the array
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < columns; j++)
            {
                Console.WriteLine("rectangularArray[{0},{1}] = {2}",
                    i, j, rectangularArray[i, j]);
            }
        }

        Console.ReadLine();
    }
}

```

توضیحاتي در مورد کد فوق :

- ۱- نحوه ي تعريف ، مقدار دهی اوليه و استفاده از آرایه های دو بعدی را در مثال فوق ملاحظه می نمایید.
 - ۲- در يك آرایه ي دوبعدی محل قرار گیری ردیف ها و ستون ها برای مثال به صورت زیر است :
- new int[rows, columns]-

استفاده از آرایه های چند بعدی :

مثال : يك برنامه ي سي شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex20 انتخاب نمایید. سپس کد زیر را درون آن بنویسید :

```

using System;

namespace ex20
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            const int rows = 4;
            const int columns = 3;
            // imply a 4x3 array
            int[,] rectangularArray =
            {
                {0,1,2},
            }
        }
    }
}

```

```

        {3,4,5},
        {6,7,8},
        {9,10,11}
    };
    for (int i = 0;i < rows;i++)
    {
        for (int j = 0;j<columns;j++)
        {
            Console.WriteLine("rectangularArray[{0},{1}] = {2}",
                                i,j,rectangularArray[i,j]);
        }
    }
}

```

توضیحاتي در مورد کد فوق :

- ۱- در حقیقت مثال فوق تعریف آرایه ایی از آرایه ها بود.
- ۲- چون مقدار دهی اولیه به صورت واضحی انجام شده نیازی به ذکر ابعاد آرایه به صورت صحیح وجود نداشت.

بحث آرایه ها ادامه دارد...

مقدمه اي بر سي شارپ : قسمت- ۱۱

Jagged arrays

Jagged arrays آرایه اي از آرایه ها است و همانطور که ذکر شد لزومی ندارد که هر ردیف آن با ردیف بعدی هم طول باشد . هنگام تعریف این نوع آرایه شما تعداد ردیف ها را مشخص می نمایید. هر ردیف يك آرایه را نگهداری می کند. در اینجا هر آرایه باید تعریف شود. روش تعریف Jagged array به صورت زیر است :

```
type [] []...
```

در اینجا تعداد براکت ها بیانگر ابعاد آرایه می باشد. برای مثال آرایه ي زیر دو بعدی است :

```
int [] [] myJaggedArray;
```

و برای مثال برای دسترسی به پنجمین عنصر آرایه ي سوم به صورت زیر عمل می شود :

```
myJaggedArray[2][4]
```

مثال : يك برنامه ي سي شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex21 انتخاب نمایید. سپس کد زیر را درون آن بنویسید :

```

using System;

namespace ex21
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)

```

```

{
    const int rows = 4;
    // declare the jagged array as 4 rows high
    int[][] jaggedArray = new int[rows][];
    // the first row has 5 elements
    jaggedArray[0] = new int[5];
    // a row with 2 elements
    jaggedArray[1] = new int[2];
    // a row with 3 elements
    jaggedArray[2] = new int[3];
    // the last row has 5 elements
    jaggedArray[3] = new int[5];
    // Fill some (but not all) elements of the rows
    jaggedArray[0][3] = 15;
    jaggedArray[1][1] = 12;
    jaggedArray[2][1] = 9;
    jaggedArray[2][2] = 99;
    jaggedArray[3][0] = 10;
    jaggedArray[3][1] = 11;
    jaggedArray[3][2] = 12;
    jaggedArray[3][3] = 13;
    jaggedArray[3][4] = 14;
    for (int i = 0; i < 5; i++)
    {
        Console.WriteLine("jaggedArray[0][{0}] = {1}",
            i, jaggedArray[0][i]);
    }
    for (int i = 0; i < 2; i++)
    {
        Console.WriteLine("jaggedArray[1][{0}] = {1}",
            i, jaggedArray[1][i]);
    }
    for (int i = 0; i < 3; i++)
    {
        Console.WriteLine("jaggedArray[2][{0}] = {1}",
            i, jaggedArray[2][i]);
    }
    for (int i = 0; i < 5; i++)
    {
        Console.WriteLine("jaggedArray[3][{0}] = {1}",
            i, jaggedArray[3][i]);
    }

    Console.ReadLine();
}
}
}

```

توضیحاتی در مورد کد فوق :

هنگام کار با آرایه های rectangular برای دسترسی به اعضا به صورت زیر عمل می شد :

rectangularArrayrectangularArray[i,j]

اما در اینجا بدین صورت است :

```
jaggedArray[3][i]
```

استفاده از System.Array :

دات نت فریم ورک کلاسی را معرفی کرده است به نام Array. توسط این کلاس کار با آرایه ها و اعمال روی آنها برای مثال سورت کردن و غیره به شدت ساده می شود .

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex22 انتخاب نمایید. سپس کد زیر را درون آن بنویسید :

```
using System;

namespace ex22
{
    class Class1
    {
        public static void PrintMyArray(object[] theArray)
        {
            foreach (object obj in theArray)
            {
                Console.WriteLine("Value: {0}", obj);
            }
            Console.WriteLine("\n");
        }

        [STAThread]
        static void Main(string[] args)
        {
            String[] myArray = {
                "Who", "is", "John", "Galt"
            };

            PrintMyArray(myArray);
            Array.Reverse(myArray);
            PrintMyArray(myArray);
            String[] myOtherArray = {
                "We", "Hold", "These", "Truths",
                "To", "Be", "Self", "Evident" };

            PrintMyArray(myOtherArray);
            Array.Sort(myOtherArray);
            PrintMyArray(myOtherArray);

            Console.ReadLine() ;

        }
    }
}
```

توضیحاتی در مورد کد فوق :

از دو متد Sort و Reverse در اینجا برای سورت کردن و نمایش آرایه به ترتیب معکوس (از انتها به ابتدا) استفاده گردیده است.

تعریف آرایه های دینامیک در سی شارپ :

یکی از مشکلاتی که با آرایه های معمول وجود دارد این است که قبل از هر کاری باید طول آنها را مشخص کرد. گاهی از اوقات ما دقیقاً نمی دانیم برنامه چه تعداد عضو را دریافت می کند تا آرایه ای از پیش تعریف شده با همان تعداد عضو ایجاد کنیم. برای حل این مشکل از کلاس ArrayList تعریف شده در دات نت فریم ورک می توان استفاده کرد.

هنگام استفاده از ArrayList نیازی به دانستن تعداد اعضایی که باید اضافه شوند نمی باشد و با استفاده از متد Add آن به سادگی می توان اعضاء را به آن اضافه نمود. تعدادی از خواص و متدهای این کلاس به صورت زیر هستند :

Adapter , FixedSize , ReadOnly , Repeat , Synchronized , Capacity, Count , IsFixedSize , IsReadOnly , IsSynchronized , Item , SyncRoot , Add , AddRange , BinarySearch , Clear , Clone , Contains , CopyTo , GetEnumerator , GetRange , IndexOf , Insert , InsertRange , LastIndexOf , Remove , RemoveAt , RemoveRange , Reverse , SetRange , Sort , ToArray , TrimToSize

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex23 انتخاب نمایید. سپس کد زیر را درون آن بنویسید :

```
using System;
using System.Collections;

namespace ex23
{
    // a simple class to store in the array
    public class Employee
    {
        public Employee(int empID)
        {
            this.empID = empID;
        }
        public override string ToString( )
        {
            return empID.ToString( );
        }
        public int EmpID
        {
            get
            {
                return empID;
            }
            set
            {
                empID = value;
            }
        }
        private int empID;
    }
}
```



```

class Class1
{
    [STAThread]
    static void Main(string[] args)
    {

        ArrayList empArray = new ArrayList( );
        ArrayList intArray = new ArrayList( );
        // populate the array
        for (int i = 0;i<5;i++)
        {
            empArray.Add(new Employee(i+100));
            intArray.Add(i*5);
        }
        // print all the contents
        for (int i = 0;i<intArray.Count;i++)
        {
            Console.Write("{0} ", intArray[i].ToString( ));
        }
        Console.WriteLine("\n");
        // print all the contents of the button array
        for (int i = 0;i<empArray.Count;i++)
        {
            Console.Write("{0} ", empArray[i].ToString( ));
        }
        Console.WriteLine("\n");
        Console.WriteLine("empArray.Capacity: {0}",
            empArray.Capacity);

        Console.ReadLine();

    }
}

```

توضیحاتی در مورد کد فوق :

- ۱- با کلمه ی کلیدی `override` در قسمت های بعدی آشنا خواهیم شد.
- ۲- برای استفاده از `ArrayList` لازم بود تا فضای نامی را که این کلاس در آن تعریف شده است ، به برنامه اضافه کرد.
- ۳- در مثال فوق نحوه ی تعریف دو کلاس را در یک فضای نام مشاهده می نمایید.
- ۴- نحوه ی تعریف و مقدار دهی `ArrayList` و همچنین استفاده از خواص آن در مثال فوق بررسی شده است.

مقدمه ای بر سی شارپ : قسمت- ۱۲

از این قسمت به بعد می خواهیم نگاهی دقیق تر به بحث شیء گرایی در سی شارپ بیاندازیم؛ همانند فضاهای نام ، کلاس ها ، ارث بری ، پلی مرفیسم و غیره.

در قسمت های قبل آشنایی مختصری با فضاهای نام پیدا کردیم. در ادامه جزئیات بیشتری را در مورد آن بررسی خواهیم کرد.

فضاهای نام (namespaces) برای اداره کردن و نظم بخشیدن به کدها ارائه شده اند. همچنین از امکان تشابه اسمی در بین قسمت های مختلف برنامه نیز جلوگیری می کنند. استفاده از آنها عادت پسندیده ای است هنگامیکه قصد داریم از کد نوشته شده بارها و بارها استفاده کنیم.

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex24 انتخاب نمایید. سپس کد زیر را درون آن بنویسید :

```
// Namespace Declaration
using System;

namespace ex24
{
    namespace tutorial
    {
        // Program start class
        class NamespaceCSS
        {
            // Main begins program execution.
            public static void Main()
            {
                // Write to console
                Console.WriteLine("This is the new Namespace.");
            }
        }
    }
}
```

توضیحاتی در مورد کد فوق :
یکی از روش های مناسب برای معرفی فضاهای نام ، ارائه ی آنها به صورت سلسله مراتبی می باشد. قسمت های عمومی تر در بالا و قسمت های اختصاصی تر در فضاهای نام داخلی تر قرار داده می شوند. این روش به معرفی فضاهای نام تو در تو منتهی می شود (nested namespaces) ، همانند مثال بالا.

کد فوق را به صورت زیر با استفاده از عملگر دات (.) می توان خلاصه نویسی کرد و نتیجه با قبل تفاوتی ندارد:

```
// Namespace Declaration
using System;

namespace ex24.tutorial
{
    // Program start class
    class NamespaceCSS
    {
        // Main begins program execution.
        public static void Main()
        {
            // Write to console
            Console.WriteLine("This is the new Namespace.");
        }
    }
}
```

طریقه ی فراخوانی اعضای فضاهای نام :

مثال : یک برنامه ی سی شارپ جدید console را در VS.NET باز کنید و نام آنرا در ابتدا ex25 انتخاب نمایید. سپس کد زیر را درون آن بنویسید :

```
// Namespace Declaration
using System;

namespace ex25
{
    // nested namespace
    namespace tutorial
    {
        class myExample1
        {
            public static void myPrint1()
            {
                Console.WriteLine("calling another namespace member1.");
            }
        }

        // Program start class
        class NamespaceCalling
        {
            // Main begins program execution.
            public static void Main()
            {
                // Write to console
                tutorial.myExample1.myPrint1();
                tutorial.myExample2.myPrint2();
            }
        }
    }

    // same namespace as nested namespace above
    namespace ex25.tutorial
    {
        class myExample2
        {
            public static void myPrint2()
            {
                Console.WriteLine("calling another namespace member2.");
            }
        }
    }
}
```

توضیحاتی در مورد کد فوق :
در کد فوق نحوه ی استفاده از اعضای تعریف شده در فضاها ی نام را می توان مشاهده کرد. نحوه ی استفاده از آنها همانطور که در قسمت های قبل نیز گفته شد به صورت زیر است :

ProjectName.Namespace.ClassName.MemberName

برای مثال در فضا ی نام tutorial کلاس myExample1 قرار دارد و داخل آن متد myPrint1 تعریف شده است. پس نحوه ی دسترسی به متد آن به صورت زیر است :

```
tutorial.myExample1.myPrint1();
```

کلاس های myExample1 و myExample2 هر دو به يك فضاي نام (ex25.tutorial) تعلق دارند ، هر چند جدا از هم نوشته شده اند. حتي آنها را با حفظ سلسله مراتب خودشان مي توان در فايلهاي جداگانه اي نيز نوشت.

استفاده از using :

مثال : يك برنامه ي سي شارپ جديد console را در VS.NET باز كنيد و نام آنرا در ابتدا ex26 انتخاب نماييد. سپس كد زير را درون آن بنويسيد :

```
// Namespace Declaration
using System;
using ex26.tutorial;

// Program start class
class UsingDirective
{
    // Main begins program execution.
    public static void Main()
    {
        // Call namespace member
        myExample.myPrint();
    }
}

// C# Namespace
namespace ex26.tutorial
{
    class myExample
    {
        public static void myPrint()
        {
            Console.WriteLine("Example of using a using directive.");
        }
    }
}
```

توضيحاتي در مورد كد فوق :

همانند مثال بالا ، براي خلاصه نويسي مي توان از كلمه ي using به همراه نام namespace مورد نظر استفاده كرد. براي مثال اگر متد WriteLine را بخواهيم كامل بنويسيم به صورت زير است :

System.Console.WriteLine(...);

اما با قيد كردن و الحاق كردن فضاي نام آن ، ديگر نيازي به ذكر System در ابتداي آن نيست.

نکته :

باز هم مي توان خلاصه نويسي بيشتري را ارائه داد

using csTut = ex26.tutorial.myExample; // alias

در این صورت تنها کافی است متد کلاس تعریف شده در آنرا به صورت زیر فراخوانی کنیم :

```
csTut.myPrint();
```

مقدمه ای بر سی شارپ : قسمت- ۱۳

کلاس ها در سی شارپ :

تا بحال در حد کاربرد ، با کلاس ها آشنا شده ایم . اما در این قسمت می خواهیم نگاهی دقیق تر به کلاس ها بیاندازیم.

هر کدی در سی شارپ قسمتی از یک کلاس می باشد و ترکیب تمام خواص و متدهای موجود در یک کلاس یک نوع داده ی جدید تعریف شده از طرف ما را پدید می آورد. هر متغیری که از کلاس ساخته شود ، شیء نامیده می شود و یک کپی منحصر به فرد است. برای مثال برنامه ی زیر را در نظر بگیرید :

```
using System;

class Data
{
    public int x;
}
class App
{
    public static void Main()
    {
        Data d1 = new Data();
        d1.x = 1;
        Data d2 = new Data();
        d2.x = 2;
        Console.WriteLine("d1.x = {0}", d1.x);
        Console.WriteLine("d2.x = {0}", d2.x);
    }
}
```

در اینجا کلاس Data تعریف شده است و دارای یک عضو به نام x می باشد. به این نوع داده در کلاس فیلد گفته می شود و هنگامیکه به صورت public معرفی می شود یعنی خارج از کلاس نیز قابل دسترسی است. در کد بالا دو متغیر از کلاس تعریف و مقدار دهی اولیه شده اند. خروجی برنامه به صورت زیر است :

```
d1.x = 1
d2.x = 2
```

دلیل این خروجی آن است که هر instance (نمونه) از کلاس منحصر بفرد است و در اینجا نمی توان انتظار داشت که هر دو خروجی یکی شوند.

برای مقدار دهی اولیه متغیرهایی که به صورت فیلد تعریف می شوند ، بهتر است مقدار دهی آنها را در سازنده ی کلاس (constructor) انجام دهیم.

```
class Data
{
    public int x;
    public Data(){x = 99;}
}
```

همانطور که پیشتر نیز ذکر شد ، متدی که هم نام کلاس است ، سازنده نام می گیرد. یک کلاس می تواند بیش از یک سازنده داشته باشد. برای مثال :

```
class Data
{
    public int x;
    private Data(){}
    public Data(int y){x = y;}
    public Data(int y, int z){x = y + z;}
}
```

از آنجائیکه که سازنده ی بدون پارامتر ذکر شده در کد فوق private تعریف شده است بنابراین خارج از کلاس دیگر قابل دسترسی نمی باشد . بنابراین کدی خارج از کلاس ، تنها می تواند از دو سازنده ی دیگر استفاده کند. برای مثال تعریف دو متغیر جدید از این کلاس به صورت زیر می باشد :

```
Data d1 = new Data(44);
Data d2 = new Data(22, 33);
```

سی شارپ به شما اجازه می دهد تا سازنده ها را در یک کلاس توسط کلمه ی کلیدی this نیز فراخوانی کنید یعنی بجای ذکر نام متد سازنده از کلمه ی this استفاده شود (در خود کلاس) .

اگر می خواهید متغیری را بین نمونه (instance) های مختلف یک کلاس به اشتراک بگذارید کلمه ی کلیدی static وارد صحنه می شود. به مثال زیر توجه کنید :

```
using System;

class Counted
{
    public static int count = 0;
    public Counted()
    {
        count++;
    }
    public int GetInstanceCount()
    {
        return count;
    }
}

class App
{
    public static void Main()
    {
        Counted d1 = new Counted();
        Console.WriteLine("current total {0}", d1.GetInstanceCount());
        Counted d2 = new Counted();
        Console.WriteLine("current total {0}", d2.GetInstanceCount());
        Console.WriteLine("total {0}", Counted.count);
    }
}
```

باید خاطر نشان کرد که متغیرهای استاتیک توسط نمونه های کلاس قابل دستیابی نیستند و فقط درون کلاس به شکل زیر می توان از آنها استفاده کرد :

<classname>.<staticmembername>

در مثال فوق دو نمونه از کلاس Counted تعریف شده است. با هر بار فراخوانی کلاس ، خودبخود سازنده اجرا شده و یک عدد به این شمارشگر استاتیک اضافه می شود. همانطور که ذکر شد، برای اینکه بتوان به این متغیر استاتیک در خارج از کد دسترسی پیدا کرد یک متد غیر استاتیک تعریف شده است.

در مثال فوق تابع `GetInstanceCount` تنها یک عدد را بر می گرداند. در برنامه نویسی شیء گرا مرسوم است که در این حالت به جای توابع از خواص استفاده شود که به اندازه ی کافی در مورد آنها در قسمت های قبل توضیح داده شد. در این صورت تعریف فوق به صورت زیر در می آید :

```
class Counted
{
    public static int x = 0;
    public Counted()
    {
        x++;
    }
    public int InstanceCount // property
    {
        get{return x;}
    }
}
```

و در این صورت قسمت بعدی کد به صورت زیر اصلاح می شود (فراخوانی خواص ، بدون ذکر پرانتزها بعد از نام آنها صورت می گیرد):

```
Counted d1 = new Counted();
Console.WriteLine("current total {0}", d1.InstanceCount);
Counted d2 = new Counted();
Console.WriteLine("current total {0}", d2.InstanceCount);
```

اگر یک خاصیت هم خواندنی و هم نوشتنی باشد به صورت زیر تعریف می شود :

```
private string name;
public string Name
{
    get{return name;}
    set{name = value;}
}
```

فیلدهای پابلیک را می توان خواند و یا تغییر داد. اگر لازم باشد تا کاربر نتواند آنها را تغییر دهد می توان از کلمه ی کلیدی `readonly` قبل از تعریف آنها استفاده کرد. مثال :

```
class Data
{
    public readonly int x = 42;
}
```

بحث کلاس ها ادامه دارد...

مقدمه اي بر سي شارپ : قسمت- ۱۴

ايندکسرها (Indexers)

با استفاده از ايندکسرها مي توان با يك کلاس همانند آرايه ها رفتار کرد. به مثال زير توجه کنيد :

```
using System;

/// <summary>
///     A simple indexer example.
/// </summary>
class IntIndexer
{
    private string[] myData;

    public IntIndexer(int size)
    {
        myData = new string[size];

        for (int i=0; i < size; i++)
        {
            myData[i] = "empty";
        }
    }

    public string this[int pos]
    {
        get
        {
            return myData[pos];
        }
        set
        {
            myData[pos] = value;
        }
    }
}

static void Main(string[] args)
{
    int size = 10;

    IntIndexer myInd = new IntIndexer(size);

    myInd[9] = "Some Value";
    myInd[3] = "Another Value";
    myInd[5] = "Any Value";

    Console.WriteLine("\nIndexer Output\n");

    for (int i=0; i < size; i++)
    {
        Console.WriteLine("myInd[{0}]: {1}", i, myInd[i]);
    }
}
```



```
}
}
```

در مثال فوق نحوه ی تعریف و استفاده از ایندکسر را می توان مشاهده کرد. کلاس IntIndexer حاوی آرایه ای به نام myData می باشد. بدلیل private بودن آن در خارج از کلاس قابل دسترسی نیست. این آرایه در سازنده ی کلاس (متد IntIndexer) با کلمه ی empty مقدار دهی اولیه شده است. عضو بعدی کلاس Indexer می باشد و با کلمه ی this کلیدی و براکتها مشخص شده ست (this[int pos]). همانطور که ملاحظه می فرمایید نحوه ی تعریف ایندکسر ها شبیه به تعریف خواص می باشد.

```
<modifier> <return type> this [argument list]
{
    get
    {
        // Get codes goes here
    }
    set
    {
        // Set codes goes here
    }
}
```

خروجی مثال فوق به صورت زیر است :

```
myInd[0]: empty
myInd[1]: empty
myInd[2]: empty
myInd[3]: Another Value
myInd[4]: empty
myInd[5]: Any Value
myInd[6]: empty
myInd[7]: empty
myInd[8]: empty
myInd[9]: Some Value
```

استفاده از اعداد صحیح روشی است متداول برای دسترسی به اعضای آرایه ها در بسیاری از زبانها اما ایندکسر ها در سی شارپ فراتر از این می رود. ایندکسر ها را می توان با پارامترهای متعددی تعریف کرد و هر پارامتر با نوعی مختلف (دقیقا همانند پارامترهای ورودی متدها). البته محدودیتی که اینجا وجود دارد در مورد نوع پارامتر ها است که تنها می تواند integers, enums, and strings باشد. بعلاوه قابلیت Overloading ایندکسر ها نیز وجود دارد. به همین جهت به آنها آرایه های هوشمند هم گفته می شود (smart arrays). مثال :

using System;

```
/// <summary>
/// Implements overloaded indexers.
/// </summary>
class OvrIndexer
{
    private string[] myData;
    private int arrSize;

    public OvrIndexer(int size)
```

```

{
    arrSize = size;
    myData = new string[size];

    for (int i=0; i < size; i++)
    {
        myData[i] = "empty";
    }
}

public string this[int pos]
{
    get
    {
        return myData[pos];
    }
    set
    {
        myData[pos] = value;
    }
}

public string this[string data]
{
    get
    {
        int count = 0;

        for (int i=0; i < arrSize; i++)
        {
            if (myData[i] == data)
            {
                count++;
            }
        }
        return count.ToString();
    }
    set
    {
        for (int i=0; i < arrSize; i++)
        {
            if (myData[i] == data)
            {
                myData[i] = value;
            }
        }
    }
}

static void Main(string[] args)
{
    int size = 10;
    OvrIndexer myInd = new OvrIndexer(size);

    myInd[9] = "Some Value";
    myInd[3] = "Another Value";
}

```

```

myInd[5] = "Any Value";

myInd["empty"] = "no value";

Console.WriteLine("\nIndexer Output\n");

for (int i=0; i < size; i++)
{
    Console.WriteLine("myInd[{0}]: {1}", i, myInd[i]);
}

Console.WriteLine("\nNumber of \"no value\" entries: {0}", myInd["no value"]);
}
}

```

در مثال فوق اولین ایندکسر با یک پارامتر از نوع اعداد صحیح تعریف شده است و در ایندکسر دوم از نوع رشته. خروجی برنامه ی فوق به صورت زیر است :

```

myInd[0]: no value
myInd[1]: no value
myInd[2]: no value
myInd[3]: Another Value
myInd[4]: no value
myInd[5]: Any Value
myInd[6]: no value
myInd[7]: no value
myInd[8]: no value
myInd[9]: Some Value

Number of "no value" entries: 7

```

نکته :

- ۱- امضای (لیست پارامترهای) ایندکسر ها در یک کلاس باید منحصر بفرد باشد .
- ۲- تعریف یک ایندکسر به صورت استاتیک مجاز نیست.

در صورت نیاز به ایندکسرهایی با پارمترهای ورودی متعدد می توان به صورت زیر عمل کرد :

```

public object this[int param1, ..., int paramN]
{
    get
    {
        // process and return some class data
    }
    set
    {
        // process and assign some class data
    }
}

```

یک مثال دیگر :

```
using System;
```

```

class IndexExample
{
    string Message;

    public static void Main()
    {
        IndexExample obj=new IndexExample("Welcome");

        /* This will access the String variable Message
           using array like notation
        */
        for(int i=0;i < obj.Length;i++)
        {
            Console.WriteLine(obj[i]);
        }
        obj[obj.Length-1]="e to C#";

        Console.WriteLine(obj.Message);
    }

    public IndexExample(string s)
    {
        Message=s;
    }

    public string this[int i]
    {
        get
        {
            if(i >= 0 && i < Message.Length)
            {
                return Message.Substring(i,1);
            }
            else
            {
                return "";
            }
        }
        set
        {
            if(i >= 0 && i < Message.Length)
            {
                Message=Message.Substring(0,i) + value + Message.Substring(i+1);
            }
        }
    }

    public int Length
    {
        get
        {
            if(Message!=null)
            {
                return Message.Length;
            }
            else
            {
                return 0;
            }
        }
    }
}

```

```

    }
}

```

مقدمه ای بر سی شارپ : قسمت- ۱۵

ارث بری (Inheritance) :

ارث بری یکی از مفاهیم اولیه ی برنامه نویسی شیء گرا می باشد. با استفاده از آن استفاده مجدد از کد موجود به نحوی مؤثر میسر می گردد و صرفه جویی قابل توجهی را در زمان برنامه نویسی پدید می آورد. به کد زیر دقت کنید :

```

using System;

public class ParentClass
{
    public ParentClass()
    {
        Console.WriteLine("Parent Constructor.");
    }
    public void print()
    {
        Console.WriteLine("I'm a Parent Class.");
    }
}

public class ChildClass : ParentClass
{
    public ChildClass()
    {
        Console.WriteLine("Child Constructor.");
    }
    public static void Main()
    {
        ChildClass child = new ChildClass();
        child.print();
    }
}

```

Output:

```

Parent Constructor.
Child Constructor.
I'm a Parent Class.

```

کد فوق از دو کلاس استفاده می کند. کلاس بالایی ParentClass و کلاس اصلی ChildClass می باشد. کاری که انجام شده است استفاده از کدهای کلاس والد ParentClass در کلاس بچه (!) ChildClass می باشد. برای اینکه ParentClass را بعنوان کلاس پایه برای ChildClass معرفی کنیم به صورت زیر عمل شد :

```

public class ChildClass : ParentClass

```

کلاس پایه با استفاده از معرفی کولون ":" ، پس از کلاس مشتق شده تعریف می شود. در سی شارپ تنها ارث بری یگانه پشتیبانی می شود. بنابراین تنها یک کلاس پایه را برای ارث بری می توان تعریف کرد.

ChildClass دقیقاً توانایی های ParentClass را دارا است. بنابراین می توان گفت ChildClass همان ParentClass است. برای مثال در کد فوق ChildClass دارای متد print نمی باشد اما آنرا از کلاس ParentClass به ارث برده است و در متد Main برنامه از آن استفاده گردیده است.

هنگام ساختن یک شیء از کلاس مشتق شده (derived) ، ابتدا یک نمونه از کلاس والد خود بخود ساخته می شود. این مورد در خروجی کد فوق هنگامی که متدهای سازنده ها روی صفحه چاپ شده اند قابل مشاهده است.

تبادل اطلاعات بین کلاس والد و کلاس فرزند :

به مثال زیر دقت کنید :

```
using System;

public class Parent
{
    string parentString;

    public Parent()
    {
        Console.WriteLine("Parent Constructor.");
    }

    public Parent(string myString)
    {
        parentString = myString;
        Console.WriteLine(parentString);
    }

    public void print()
    {
        Console.WriteLine("I'm a Parent Class.");
    }
}

public class Child : Parent
{
    public Child() : base("From Derived")
    {
        Console.WriteLine("Child Constructor.");
    }

    public void print()
    {
        base.print();
        Console.WriteLine("I'm a Child Class.");
    }

    public static void Main()
    {

```

```

Child child = new Child();
child.print();
((Parent)child).print();
}
}

```

Output:

```

From Derived
Child Constructor.
I'm a Parent Class.
I'm a Child Class.
I'm a Parent Class.

```

کلاس فرزند با کلاس والد در هنگام instantiation می تواند تبادل اطلاعات کند. همانطور که در مثال فوق بارز است با استفاده از کلمه ی کلیدی base ، کلاس فرزند تابع سازنده ی کلاس والد را فراخوانی کرده است. اولین خط خروجی بیانگر این موضوع است.

گاهی از اوقات ما می خواهیم تابعی را که در کلاس والد تعریف شده است را در کلاس فرزند با تعریف دیگری و مخصوص به خودمان ارائه دهیم. در اینصورت تابع تعریف شده در کلاس فرزند ، تابع هم نام والد را مخفی خواهد کرد و دیگر آن تابع والد فراخوانی نخواهد گردید. در این حالت تنها یک راه برای دسترسی به تابع اصلی والد وجود دارد و آن استفاده از base می باشد که در کد فوق پیاده سازی شده است.

با استفاده از base می توان به تمام اعضای public و یا protected کلاس والد از درون کلاس فرزند دسترسی داشت.

راه دیگری که برای این منظور وجود دارد در آخرین خط کد فوق در متد Main پیاده سازی شده است :

```
((Parent)child).print();
```

برای تبدیل نوع های مختلف در سی شارپ می توان از پرانتز و سپس ذکر نوع اصلی استفاده کرد به این عمل casting و یا boxing هم می گویند. در کد فوق درحقیقت child به نوعی از parent تبدیل شده است. بنابراین مانند این است که یک نمونه از کلاس والد متد print همان کلاس را فراخوانی می کند.

مقدمه ای بر سی شارپ : قسمت- ۱۶

پلی مرفیسم (Polymorphism)

یکی دیگر از مفاهیم اولیه ی شیء گرایی پلی مرفیسم (چند ریختی) می باشد. پلی مرفیسم به معنای توانایی استفاده کردن از فرم های مختلف یک نوع است بدون توجه به جزئیات آن . برای مثال هنگامیکه سیگنال تلفنی شما فرستاده می شود ، از نوع تلفنی که در انتهای خط موجود است خبری ندارد. تلفن انتهای خط ، می خواهد یکی از تلفن های عهد عتیق باشد و یا تلفنی با آخرین امکانات روز . شرکت مخابرات (!) تنها از نوع پایه ای به نام phone خبر دارد و فرض می کند که هر instance از این نوع می داند که چگونه صدای زنگ تلفن شما را به صدا در آورد. بنابراین شرکت مخابرات از تلفن شما به صورت پلی مرف استفاده می کند.

در عمل پلی مرفیسم هنگامی مفید خواهد بود که بخواهیم گروهی از اشیاء را به یک آرایه نسبت دهیم و سپس متدهای هر یک را فراخوانی کنیم. الزاما این اشیاء از یک نوع نخواهند بود.

نحوه ی ایجاد متدهای پلی مرفیک :

برای ایجاد متدی که نیاز است تا پلی مرفیسم را پشتیبانی نماید ، تنها کافی است آنرا از نوع virtual در کلاس پایه تعریف کنیم. مثال :

فرض کنید تابع DrawWindow در کلاس Window تعریف شده است. برای ایجاد قابلیت پلی مرفیسم در آن به صورت زیر عمل می شود :

```
public virtual void DrawWindow( )
```

در این حالت هر کلاسی که از Window مشتق شود ، مجاز است نگارش خاص خودش را از DrawWindow ارائه کند. در این صورت در کلاسی که از کلاس پایه ی ما ارث می برد ، تنها کافی است که کلمه ی کلیدی override را قبل از نام تابع مذکور ذکر نماییم.

یک مثال کامل :

```
using System;
```

```
public class DrawingObject
{
    public virtual void Draw()
    {
        Console.WriteLine("I'm just a generic drawing object.");
    }
}
```

```
public class Line : DrawingObject
{
    public override void Draw()
    {
        Console.WriteLine("I'm a Line.");
    }
}
```

```
public class Circle : DrawingObject
{
    public override void Draw()
    {
        Console.WriteLine("I'm a Circle.");
    }
}
```

```
public class Square : DrawingObject
{
    public override void Draw()
    {
        Console.WriteLine("I'm a Square.");
    }
}
```

```
public class DrawDemo
{
    public static int Main(string[] args)
    {
        DrawingObject[] dObj = new DrawingObject[4];

        dObj[0] = new Line();
    }
}
```



```

dObj[1] = new Circle();
dObj[2] = new Square();
dObj[3] = new DrawingObject();

foreach (DrawingObject drawObj in dObj)
{
    drawObj.Draw();
}

return 0;
}
}

```

کلاس DrawingObject ، کلاسی پایه برای تمام کد ما که از آن به ارث می برد ، می باشد. متد Draw در آن با کلمه ی کلیدی virtual معرفی شده است. یعنی تمام کلاس های فرزند این کلاس والد می توانند این متد را override کنند (تحریف کردن و یا تحت الشعاع قرار دادن هم ترجمه شده است!). در ادامه سه کلاس تعریف شده اند که تمامی آنها از کلاس مبنا ارث می برند و تابع Draw را تحریف کرده اند (!). با استفاده از کلمه ی کلیدی override می توان تابع مجازی کلاس مبنا را با تعریفی جدید در زمان اجرای برنامه ارائه داد. تحریف شدن تنها زمانی رخ می دهد که کلاس ، توسط ریفرنس کلاس مبنا مورد ارجاع واقع شده باشد.

و در متد Main برنامه از این کلاس ها در عمل استفاده گردیده است. در متد Main ، آرایه ای از نوع DrawingObject تعریف و مقدار دهی اولیه شده است تا بتواند ۴ شیء از نوع این کلاس را در خودش ذخیره کند.

بدلیل رابطه ی ارث بری موجود می توان آرایه ی dObj را با نوع هایی از کلاس های Line ، Circle و Square مقدار دهی کرد (همانند کدهای بعدی متد Main) . اگر ارث بری در اینجا وجود نمی داشت می بایست به ازای هر کلاس یک آرایه تعریف می شد.

سپس از حلقه ی زیبای foreach برای حرکت در بین اعضای این آرایه استفاده گردیده است. در اینجا هر شیء متد خاص خودش را در مورد Draw فراخوانی می کند و نتیجه را روی صفحه نمایش خواهد داد. خروجی نهایی به صورت زیر خواهد بود :

Output:

```

I'm a Line.
I'm a Circle.
I'm a Square.
I'm just a generic drawing object.

```

مقدمه ای بر سی شارپ : قسمت- ۱۷

کلاس های abstract

کلاس ها را همچنین می توان به صورت abstract تعریف کرد. از این نوع کلاس ها نمی توان instance ایی را ایجاد نمود. در این کلاس های پایه ، صرفاً تعریف متدها و خواص هایی عنوان گردیده و در آینده در کلاس های فرزند توسعه داده خواهند شد. برای مثال :

```
public abstract class Named
{
    public abstract String Name {get; set;} // property
    public abstract void PrintName(); // method
}
public class B : Named
{
    private String name = "empty";
    public override String Name
    {
        get{return name;}
        set{name=value;}
    }
    public override void PrintName()
    {
        Console.WriteLine("Name is {0}", name);
    }
}
```

سوالی که شاید پیش بیاید این است که اگر interface ها صرفاً تعریف توابع و خواص را می توانند در خود جای دهند پس چه دلیلی برای بکار بردن آنها و طولانی کردن کار کد نویسی وجود دارد؟ کاربردهای زیادی را می توان برای اینترفیس ها برشمرد. اینترفیس یک رفتار را تعریف می کند. فرض کنید در حال توسعه ی برنامه ای هستید که بر روی دو کامپیوتر مختلف باید با هم در ارتباط مستقیم بوده و برهم کنش داشته باشند و هر برنامه از ماژولی به نام CCommObj communication object استفاده می نماید. یکی از متدهای این شیء ، SendData() می باشد که رشته ای را دریافت کرده و به برنامه ی دیگر می فرستد. این فراخوانی از نوع asynchronous است زیرا ما نمی خواهیم اگر خطایی در شبکه رخ داد، برنامه برای همیشه منتظر باقی بماند. اما چگونه برنامه ی A که تابع ذکر شده را فراخوانی کرده است می تواند تشخیص دهد که پیغام به مقصد رسیده است یا خیر و یا آیا خطایی در شبکه مانع رسیدن پیغام گشته است یا خیر؟ جواب بدین صورت است که CCommObj هنگام دریافت پیغام ، رخدادی را سبب خواهد شد و اگر خطایی رخ داده باشد خیر. در این حالت نیاز به یک ماژول logging نیز احساس می گردد تا خطاهای رخ داده را ثبت نماید. یک روش انجام آن این است که CCommObj پیاده سازی این امکان را نیز بعهد گرفته و اگر فردا نیز خواستیم ماژول دیگری را به برنامه اضافه کنیم هر روز باید CCommObj را تغییر دهیم. تمام این کارها را به سادگی می توان در یک اینترفیس مدل کرد. روش آن نیز در ادامه بیان می گردد:

در ابتدا یک اینترفیس ایجاد می کنیم تا لیست تمام امکانات ممکن را "منتشر" کند:

```
interface ICommObjEvents
{
    void OnDataSent();
    void OnError();
}
```

شیء CCommObj ما از این توابع که بعداً توسعه داده خواهند شد برای با خبر سازی کلاپنت ها استفاده می نماید. تمام متدها در یک اینترفیس ذاتاً پابلیک هستند بنابراین نیازی به ذکر صریح این مطلب نمی باشد و اگر اینکار را انجام دهید کامپایلر خطای زیر را گوشزد خواهد کرد :

The modifier 'public' is not valid for this item

در ادامه کلاپنت CClientApp_A را پیاده سازی خواهیم کرد :

```
class CClientApp_A:ICommObjEvents
```

```

{
    public void OnDataSent()
    {
        Console.WriteLine("OnDataSent");
    }
    public void OnError()
    {
        Console.WriteLine("OnError");
    }
    private CCommObj m_Server;
    public void Init(CCommObj theSource)
    {
        m_Server = theSource;
        theSource.Advise (this);
        string strAdd = ("N450:1");
        m_Server.read (strAdd,10);
    }
}

```

در کد فوق کلاس CClientApp_A از ICommObjEvents ارث برده و تمام متدهای این اینترفیس را پیاده سازی نموده است. هنگامی که CCommObj تابع OnDataSent را فراخوانی می کند این کلاینت پیغام را دریافت خواهد کرد. لازم به ذکر است که کلاس کلاینت ما چون از یک اینترفیس ارث بری می نماید پس باید تمام توابع و خواص کلاس پایه را پیاده سازی کند در غیر اینصورت هر چند برنامه کامپایل خواهد شد اما هنگامی که شیء CCommObj هر کدام از توابع این کلاس را فراخوانی کند ، خطای زمان اجرا رخ خواهد داد. متد Init کلاس فوق آرگومانی را از نوع CCommObj دریافت نموده و در یک متغیر private آنرا ذخیره می نماید. همچنین در این متد ، متد Advise از کلاس CCommObj نیز فراخوانی گشته است.

```

public class CCommObj
{
    private int m_nIndex;
    public ICommObjEvents [] m_arSinkColl;
    public CCommObj()
    {
        m_arSinkColl = new ICommObjEvents[10];
        m_nIndex = 0;
    }
    public int Advise(ICommObjEvents theSink)
    {
        m_arSinkColl[m_nIndex] = theSink;
        int lCookie = m_nIndex;
        m_nIndex++;
        return lCookie
    }
    public void SendData(string strData)
    {
        foreach ( ICommObjEvents theSink in m_arSinkColl)
            if(theSink != null )
                theSink.OnDataSent ();
    }
}

```

این بحث ادامه دارد (لطفاً با ما باشید و جایی نروید چون تازه بحث شيء گرایي شروع شده است!!)

مقدمه ای بر سی شارپ : قسمت- ۱۸

در کلاس CCommObj که با آن آشنا شدیم ، آرایه ای Private از نوع ICommObjEvents به نام m_arSinkColl وجود دارد. این آرایه تمام اینترفیس های sink شده را ذخیره می کند. واژه ی sink در اینجا به کلاسی گفته می شود که دریافت کننده ی رخدادها است. متد Advise تنها sink وارده به آنرا در يك آرایه ذخیره می کند و سپس اندیس آرایه را که در اینجا cookie نامیده شده است بر می گرداند. این کوکی توسط کلاينتي که دیگر نمی خواهد از آن آیتم هیچونه رخدادی را دریافت کند به سرور فرستاده می شود و سپس سرور این آیتم را از لیست خودش حذف خواهد کرد.

نحوه ی فراخوانی متد advise توسط کلاينت نیز جالب است.

```
public void Init(CCommObj theSource)
{
    m_Server = theSource;
    theSource.Advise (this);
    string strAdd = ("Hello");
    m_Server.read (strAdd,10);
}
```

در اینجا تنها يك this بعنوان آرگومان به متد advice فرستاده شده است در حالیکه انتظار می رفت آرگومانی از نوع **ICommObjEvents** به تابع فرستاده شود. دلیل صحت این عمل بدین صورت است که کلاس ClientApp_A از اینترفیس ICommObjEvents ارث برده است و آنرا پیاده سازی نموده است.

در ادامه لیست کامل برنامه ی نوشته شده را در حالت Console ملاحظه می فرمایید.

```
namespace CSharpCenter
{
    using System;

    public interface ICommObjEvents
    {
        void OnDataSent();
        void OnError();
    }
    public class CCommObj
    {
        private int m_nIndex;
        public ICommObjEvents [] m_arSinkColl;
        public CCommObj()
        {
            m_arSinkColl = new ICommObjEvents[10];
            m_nIndex = 0;
        }

        public void Advise(ICommObjEvents theSink)
        {
```

```

        m_arSinkColl[m_nIndex] = theSink;
        m_nIndex++;
    }
    public void SendData(string strData)
    {
        foreach ( ICommObjEvents theSink in m_arSinkColl)
        {
            if(theSink != null )
            {
                theSink.OnDataSent ();
            }
        }
    }
}
class CClientApp_A:ICommObjEvents
{
    public void OnDataSent()
    {
        Console.WriteLine("OnDataSent Client App A");
    }
    public void OnError()
    {
        Console.WriteLine("OnError");
    }
    public void Read()
    {
        string strAdd = ("Hello");
        m_Server.SendData (strAdd);
    }
    private CCommObj m_Server;
    public void Init(CCommObj theSource)
    {
        m_Server = theSource;
        theSource.Advise (this);
    }
}
class CClientApp_B:ICommObjEvents
{
    public void OnDataSent()
    {
        Console.WriteLine("OnDataSent Client App B");
    }
    public void OnError()
    {
        Console.WriteLine("OnError");
    }
    private CCommObj m_Server;
    public void Init(CCommObj theSource)
    {
        m_Server = theSource;
        theSource.Advise (this);
    }
}
class ConsoleApp

```

```

{
    public static void Main()
    {
        CClientApp_A theClient = new CClientApp_A ();
        CClientApp_B theClient2 = new CClientApp_B ();
        CCommObj theComm = new CCommObj ();
        theClient.Init (theComm);
        theClient2.Init (theComm);
        theClient.Read();
    }
}

```

در متد Main برنامه ی فوق ، ما دو کلاينت تعريف کرده ایم و يك نمونه از CCommObj را. دو کلاينت instance هاي CCommObj را بعنوان آرگومان دریافت کرده اند. در هنگام فراخواني init توسط هر کلاينت متد advise فراخواني مي گردد. در خاتمه Read مربوط به کلاينت ۱ فراخواني شده است که سبب مي شود تا رخداد OnDataSend شيء CCommObj اجرا شود و به تمام کلاينت ها فرستاده شود.

هدف از این مثال ارائه ی بعضي از جنبه هاي اینترفیس ها و نحوه ی استفاده از آنها بود. دو مطلب دیگر در مورد اینترفیس ها باقي مانده اند تا به پایان بحث مربوط به آنها برسیم:

چگونه مي توان متوجه شد که يك شيء واقعا يك اینترفیس را پیاده سازي کرده است؟
 دو روش براي فهمیدن این موضوع وجود دارد:
 - استفاده از کلمه ی کلیدی is
 - استفاده از کلمه ی کلیدی as

اولین مثال زیر از کلمه ی کلیدی is استفاده مي کند :

```

CClientApp_C theClient3 = new CClientApp_C ();
if(theClient3 is ICommObjEvents)
    Console.WriteLine ("theClient3 implements ICommObjEvents");
else
    Console.WriteLine ("theClient3 doesnot implement ICommObjEvents");

```

کلمه ی کلیدی is مقدار true را بر مي گرداند اگر اپراتور سمت چپ ، اینترفیس سمت راست را پیاده سازي کرده باشد.

```

ICommObjEvents theClient5 = theClient3 as ICommObjEvents;
if(theClient5 != null )
    Console.WriteLine ("Yes theClient implements interface");
else
    Console.WriteLine ("NO,Yes theClient doesn't implements interface");

```

در مثال فوق اپراتور as در حال casting شيء theClient5 به ICommObjEvents مي باشد. چون CClientApp_C اینترفیس را پیاده سازي نمي کند حاصل خط اول نال خواهد بود.

به صورت خلاصه :
یک اینترفیس قراردادی است که به کلاینت گارانتی می دهد یک کلاس خاص چگونه رفتار خواهد کرد. هنگامیکه کلاسی یک اینترفیس را پیاده سازی می کند به تمام کلاینت ها می گوید که : من تمام موارد ذکر شده در اینترفیس را ارائه و پیاده سازی خواهم کرد. نمونه ی عملی استفاده از اینترفیس ها بحث dot net remoting است.

مقدمه ای بر سی شارپ : قسمت- ۱۹

مقابله با خطاها در سی شارپ (Exception Handling in C#)

EXCEPTION یک خطای زمان اجرا است که بدلیل شرایطی غیرنرمال در برنامه ایجاد می شود. در سی شارپ exception کلاسی است در فضای نام سیستم. شیء آیی از نوع exception بیانگر شرایطی است که سبب رخ دادن خطا در کد شده است. سی شارپ از exception ها به صورتی بسیار شبیه به جاوا و سی پلاس پلاس استفاده می نماید.

- دلایلی که باید در برنامه exception handling حتما صورت گیرد به شرح زیر است:
- قابل صرفنظر کردن نیستند و اگر کدی این موضوع را در نظر نگیرد با یک خطای زمان اجرا خاتمه پیدا خواهد کرد.
- سبب مشخص شدن خطا در یک نقطه از برنامه شده و ما را به اصلاح آن سوق می دهد.

بوسیله ی عبارات try...catch می توان مدیریت خطاها را انجام داد. کدی که احتمال دارد خطایی در آن رخ دهد درون try قرار گرفته و سپس بوسیله ی یک یا چند قطعه ی catch می توان آنرا مدیریت کرد. و اگر از این قطعات خطایابی استفاده نشود برنامه به صورتهای زیر متوقف خواهد شد :

```
class A {static void Main() {catch {}}}  
TEMP.cs(3,5): error CS1003: Syntax error, 'try' expected  
  
class A {static void Main() {finally {}}}  
TEMP.cs(3,5): error CS1003: Syntax error, 'try' expected  
  
class A {static void Main() {try {}}}  
TEMP.cs(6,3): error CS1524: Expected catch or finally
```

بهتر است یک مثال ساده را در این زمینه مرور کنیم:

```
int a, b = 0 ;  
Console.WriteLine( "My program starts " ) ;  
try  
{  
    a = 10 / b;  
}  
catch ( Exception e )  
{  
    Console.WriteLine ( e ) ;  
}  
Console.WriteLine ( "Remaining program" ) ;
```

The output of the program is:

```
My program starts
System.DivideByZeroException: Attempted to divide by zero.
   at ConsoleApplication4.Class1.Main(String[] args) in
   d:\dont delete\consoleapplication4\class1.cs:line 51
Remaining program
```

برنامه شروع به اجرا می کند. سپس وارد بلوک و یا قطعه ی try می گردد. اگر هیچ خطایی هنگام اجرای دستورات داخل آن رخ ندهد ، برنامه به خط آخر جهش خواهد کرد و کاری به قطعات catch ندارد. اما در اینجا در اولین try عددی بر صفر تقسیم شده است بنابراین کنترل برنامه به بلوک catch منتقل می شود و صرفاً نوع خطای رخ داده شده نوشته و نمایش داده می شود. سپس برنامه به کار عادی خودش ادامه می دهد.

تعدادی از کلاس های exception در سی شارپ که از کلاس System.Exception ارث برده اند به شرح زیر هستند :

- Exception Class - - Cause
- SystemException - A failed run-time check;used as a base class for other.
- AccessException - Failure to access a type member, such as a method or field.
- ArgumentException - An argument to a method was invalid.
- ArgumentNullException - A null argument was passed to a method that doesn't accept it.
- ArgumentOutOfRangeException - Argument value is out of range.
- ArithmeticException - Arithmetic over - or underflow has occurred.
- ArrayTypeMismatchException - Attempt to store the wrong type of object in an array.
- BadImageFormatException - Image is in the wrong format.
- CoreException - Base class for exceptions thrown by the runtime.
- DivideByZeroException - An attempt was made to divide by zero.
- FormatException - The format of an argument is wrong.
- IndexOutOfRangeException - An array index is out of bounds.
- InvalidCastException - An attempt was made to cast to an invalid class.
- InvalidOperationException - A method was called at an invalid time.
- MissingMemberException - An invalid version of a DLL was accessed.
- NotFiniteNumberException - A number is not valid.
- NotSupportedException - Indicates sthat a method is not implemented by a class.
- NullReferenceException - Attempt to use an unassigned reference.
- OutOfMemoryException - Not enough memory to continue execution.
- StackOverflowException - A stack has overflowed.

در کد فوق صرفاً عمومی ترین نوع از این کلاس ها که شامل تمامی این موارد می شود مورد استفاده قرار گرفت یعنی :

```
catch ( Exception e )
```

اگر نیازی به خطایابی دقیقتر باشد می توان از کلاس های فوق برای اهداف مورد نظر استفاده نمود.

مثالی دیگر: (در این مثال خطایابی دقیق تر با استفاده از کلاس های فوق و همچنین مفهوم finally نیز گنجانده شده است)


```

int a, b = 0 ;
Console.WriteLine( "My program starts" ) ;
try
{
    a = 10 / b;
}
catch ( InvalidOperationException e )
{
    Console.WriteLine ( e ) ;
}
catch ( DivideByZeroException e)
{
    Console.WriteLine ( e ) ;
}
finally
{
    Console.WriteLine ( "finally" ) ;
}
Console.WriteLine ( "Remaining program" ) ;

```

The output here is:

```

My program starts
System.DivideByZeroException: Attempted to divide by zero.
at ConsoleApplication4.Class1.Main(String[] args) in
d:\dont delete\consoleapplication4\class1.cs:line 51
finally
Remaining program

```

قسمت موجود در قطعه ي فاينالي همواره صرفنظر از قسمت هاي ديگر اجرا مي شود.

به مثال زير دقت كنيد :

```

int a, b = 0 ;
Console.WriteLine( "My program starts" )
try
{
    a = 10 / b;
}
finally
{
    Console.WriteLine ( "finally" ) ;
}
Console.WriteLine ( "Remaining program" ) ;

```

Here the output is

```

My program starts
Exception occurred: System.DivideByZeroException:
    Attempted to divide by zero.at ConsoleApplication4.Class1.
    Main(String[] args) in d:\dont delete\consoleapplication4
    \class1.cs:line 51
finally

```

قسمت چاپ Remaining program اجرا نشده است.

عبارت throw :

این عبارت سبب ایجاد يك خطا در برنامه مي شود.

مثال :

```
int a, b = 0 ;
Console.WriteLine( "My program starts" ) ;
try
{
    a = 10 / b;
}
catch ( Exception e)
{
    throw
}
finally
{
    Console.WriteLine ( "finally" ) ;
}
```

در این حالت قسمت فاینالی اجرا شده و برنامه بلافاصله خاتمه پیدا مي کند.

مقدمه اي بر سي شارپ : قسمت- ۲۰

سربارگذاري عملگر ها (Operator OverLoading)

به تعريف مجدد راه و روش اجراي عملگر ها توسط ما ، سربارگذاري عملگرها گفته مي شود. فرض كنيد مي خواهيد عدد ۲ را به يك مقدار datetime اضافه كنيد. خطاي زير حاصل خواهد شد:

CS0019: Operator '+' cannot be applied to operands of type 'System.DateTime' and 'int'

جالب بود اگر مي توانستيم عدد ۲ را به datetime اضافه كنيم و نتيجه ي آن تعداد روزهاي مشخص بعلاوه ي دو مي بود. اينگونه توانايي ها را مي توان بوسيله ي operator overloading ایجاد کرد.

تنها عملگر هاي زير را مي توان overload کرد:

Unary Operators

+ - ! ~ ++ -- true false

Binary Operators

+ - * / % & | ^ << >> == != > < >= <=

نحوه ي انجام اينكار نيز در حالت كلي به صورت زير است:

return_datatype operator operator_to_be_overloaded (agruments)

```
{
}
```

به مثال زیر توجه کنید:

```
using System;
class MyDate
{
    public DateTime tempDate;
    public MyDate(int year,int month,int day)
    {
        tempDate=new DateTime(year,month,day);
    }
    public static DateTime operator + (MyDate D,int noOfDays)
    {
        return D.tempDate.AddDays(noOfDays);
    }
    public static DateTime operator + (int noOfDays,MyDate D)
    {
        return D.tempDate.AddDays(noOfDays);
    }
}

class Test
{
    static void Main()
    {
        MyDate MD=new MyDate(2001,7,16);
        Console.WriteLine(MD + 10 );
    }
}
```

output:
2001-07-26

در مثال فوق عملگر + دوبار overload شده است. یکبار توسط آن می توان یک عدد صحیح را به یک تاریخ اضافه کرد و بار دیگر یک یک تاریخ را می توان به عدد صحیح افزود.

موارد زیر را هنگام سربارگذاری عملگرها به خاطر داشته باشید:

- ۱- تنها اپراتورهای ذکر شده را می توان overload کرد. اپراتورهایی مانند **new,typeof, sizeof** و غیره را نمی توان سربارگذاری نمود.
- ۲- خروجی متدهای بکار گرفته شده در سربارگذاری عملگرها نمی تواند **void** باشد.
- ۳- حداقل یکی از آرگومانهای بکار گرفته شده در متدی که برای overloading عملگرها بکار می رود باید از نوع کلاس حاوی متد باشد.
- ۴- متدهای مربوطه باید به صورت **public** و **static** تعریف شوند.
- ۵- هنگامی که اپراتور < را سربارگذاری می کنید باید جفت متناظر آن یعنی > را هم سربارگذاری نمایید.
- ۶- هنگامیکه برای مثال + را overload می کنید خودبخود += نیز overload شده است و نیازی به کدنویسی برای آن نیست.

یکی از موارد جالب بکار گیری سربارگذاری عملگرها در برنامه نویسی سه بعدی و ساختن کلاسی برای انجام عملیات ماتریسی و برداری می باشد.

مقدمه ای بر سی شارپ : قسمت- ۲۱

Delegates

Delegates در سی شارپ روشی مطمئن و typesafe را برای بکار گیری مفهوم function pointer ارائه می دهند. یکی از ابتدایی ترین استفاده های function pointers پیاده سازی callback می باشد. اما در ابتدا لازم است تا با اصول اولیه ی کاری آن آشنا شویم.

مثال یک :

یک delegate چگونه تعریف و استفاده می شود؟

Delegate یک شیء است که بیانگر یک تابع می باشد بنابراین می تواند بعنوان آرگومان ورودی یک تابع دیگر و یا عضوی از یک کلاس بکار رود.

در زبان "function-pointer" ، Func1() اشاره گری به Func2() را بعنوان پارامتر دریافت کرده و نهایتاً آنرا فراخوانی می کند.

در زبان "delegate" ، Func1() یک شیء delegate از Func2() را دریافت کرده و سپس آنرا فراخوانی می کند.

در مثال زیر از دو تابع برای شرح این مطلب سود جسته شده است:

Func1() از delegate استفاده می کند.

Func2() یک delegate است.

(شماره گذاری خطوط ، در کد زیر ، صرفاً برای راحت تر شدن توضیحات در مورد آنها است و لزومی به تایپ آنها در برنامه ی اصلی نیست.)

```
01 using System;
02 delegate void Delg(string sTry);
03 public class Example1{

    // function which uses the delegate object
04 private static void Func1(Delg d){
05     d("Passed from Func1");
06 }

    // function which is passed as an object
07 private static void Func2(string sToPrint){
08     Console.WriteLine("{0}",sToPrint);
09 }

    // Main execution starts here
10 public static void Main(){
11     Delg d = new Delg(Func2);
12     Func1(d);
13 }
14 }
```

LINE 02

یک شیء delegate را برای Func2 تعریف می کند.

LINE 04-06

تابعی را تعریف کرده است که آرگومان ورودی آن از نوع Delg است.

LINE 07-09

تابعی را تعریف می کند که باید به صورت delegate به تابع دیگر فرستاده شود.

LINE 10-14

تابع Main اجرای برنامه را با ایجاد یک شیء delegate برای Func2 آغاز کرده و سپس تابع Func1 را فراخوانی می کند.

مثال ۲:

چگونه می توان از delegates در کارهای عملی استفاده کرد؟

طرح یک مساله:

شخصی تقاضای ثبت نام در یک مؤسسه ی آموزشی و همچنین تقاضای کاربایی در یک شرکت را داده است. هر کدام از این نهادها روشی خاص خود را برای ارزیابی شخص دارند.

راه حل (با روشی شیء گرا):

شخص مشخصاتی همچون سن / جنس / میزان تحصیلات قبلی / تجربیات کاری و مدارک مرتبط دارد. مؤسسه ی آموزشی تعدادی از این مشخصات را برای ارزیابی شخص استفاده می کند و این امر در مورد شرکت یاد شده نیز صادق است. شیء شرکت و شیء آموزشگاه هر کدام توابع ارزیابی خاص خودشان را پیاده سازی می کنند. شخص ، اینترفیسی واحدی را در اختیار شرکت / آموزشگاه برای ارزیابی خود قرار می دهد.

پیاده سازی (با استفاده از سی شارپ):

ما delegateای را تعریف می کنیم که بیانگر اینترفیسی است که به شرکت و آموزشگاه اجازه ی چک کردن شخص را می دهد.

سه کلاس school و company و person را تعریف می نماییم. کلاس test را برای آزمودن این موارد ایجاد می کنیم.

```
01 using System;
02 using System.Collections;

03 public delegate bool GetChecker(Person p);

    // Person has his information with him as he
    // applies for School and Company
04 public class Person
05 {
06     public string Name;
07     public int Age;
08     public bool Graduate;
09     public int YearsOfExp;
10     public bool Certified;

11     public Person(string name,
12                     int age,
13                     bool graduate,
14                     int yearsOfExp,
15                     bool certified)
16     {
17         Name=name;
18         Age=age;
19         Graduate=graduate;
20         YearsOfExp=yearsOfExp;
21         Certified=certified;
22     }
23 }
```

```

19 public bool CheckMe(GetChecker checker)
20 {
21     return(checker(this));
22 }
23 }

// A school, the person applied for higher studies
24 public class School
25 {
26     public static bool SchoolCheck(Person p)
27     {
28         return (p.Age>10 && p.Graduate);
29     }
30 }

// A Company, the person wants to work for
31 public class Company
32 {
33     public static bool CompanyCheck(Person p)
34     {
35         return (p.YearsOfExp>5 && p.Certified);
36     }
37 }

// A Test class, displays delegation in action
38 public class Test
39 {
40     public static void Main()
41     {
42         Person p1 = new Person("Jack",20,true,6,false);
43         Console.WriteLine("{0} School Check : {1}",
                                p1.Name,
                                p1.CheckMe(new GetChecker(School.SchoolCheck)));
44         Console.WriteLine("{0} Company Check : {1}",
                                p1.Name,
                                p1.CheckMe(new GetChecker(Company.CompanyCheck)));
45     }
46 }

```

LINE 03

Delegate مورد نیاز را تعریف می کند.

LINE 04-23

کلاس person را تعریف می کند. این کلاس تابعی پابلیک را ارائه می دهد که آرگومان ورودی آن از نوع GetChecker می باشد.

LINE 24-30

کلاس school را تعریف می کند و سپس تابعی را که delegate است ارائه می دهد.

LINE 31-37

کلاس company را تعریف می کند و سپس تابعی را که delegate است ارائه می دهد.

LINE 38-36

كلاس test را پياده سازي مي نمايد. سپس يك شيء شخص ساخته مي شود. در ادامه new GetChecker(School.SchoolCheck) و new GetChecker(Company.CompanyCheck) شيء ايي را ايجاد مي كند از نوع delegate مورد نياز و آنرا به تابع CheckMe مي فرستد. خروجي نتيجه ي ارزيابي اين شخص مي باشد.

اگر چك كردن اشخاص بيشتر نياز باشد به اين صورت عمل مي شود:

```
Person p1 = new Person("Jack",20,true,6,false);
Person p2 = new Person("Daniel",25,true,10,true);
GetChecker checker1= new GetChecker(School.SchoolCheck);
GetChecker checker2= new GetChecker(School.CompanyCheck);

Console.WriteLine("{0} School Check : {1}",
    p1.Name,p1.CheckMe(checker1));
Console.WriteLine("{0} Company Check : {1}",
    p1.Name,p1.CheckMe(checker2));
Console.WriteLine("{0} School Check : {1}",
    p2.Name,p2.CheckMe(checker1));
Console.WriteLine("{0} Company Check : {1}",
    p2.Name,p2.CheckMe(checker2));
```

مثال ۳ :

Delegates در تعامل بين دات نت فریم ورك و سي شارپ چه نقشي دارد؟

طرح يك مساله:

نمايش دادن ميزان پيشرفت خواندن يك فايل هنگامي كه حجم فايل بسيار زياد است.

راه حل (با استفاده از سي شارپ):

در مثال زير از كلاس FileReader براي خواندن يك فايل حجيم استفاده شده است. هنگامي كه برنامه مشغول خواندن فايل است 'Still reading..' را نمايش مي دهد و در پايان 'Finished reading..' را عرضه مي كند. براي اينكار از فضاي نام System.IO استفاده شده است. اين فضاي نام حاوي delegate ايي مهيا شده براي ما مي باشد. بدین ترتيب مي توانيم به دات نت فریم ورك بگويم كه ما تابعي را تعريف كرده ايم كه او مي تواند آنرا فراخواني كند.

سؤال: چه نيازي وجود دارد تا دات نت فریم ورك تابع ما را فراخواني و اجرا كند؟ با استفاده از تابع ما كه دات نت فریم آنرا صدا خواهد زد در طول خواندن فايل به ما گفته مي شود كه بله! من هنوز مشغول خواندن هستم! به اين عمليات Callback نيز گفته مي شود. به اينكار پردازش asynchronous نيز مي گويند!

```
01 using System;
02 using System.IO;

03 public class FileReader{
04     private Stream sInput;
05     private byte[] arrByte;
06     private AsyncCallback callbackOnFinish;

07     public FileReader(){
08         arrByte=new byte[256];
09         callbackOnFinish = new AsyncCallback(this.readFinished);
10     }

11     public void readFinished(IAsyncResult result){
```

```

12     if (sInput.EndRead(result)>0) {
13         sInput.BeginRead(arrByte,
                            0,
                            arrByte.Length,
                            callbackOnFinish,
                            null);
14         Console.WriteLine("Still reading..");
15     }
16     else Console.WriteLine("Finished reading..");
17 }

18 public void readFile() {
19     sInput = File.OpenRead(@"C:\big.dat");
20     sInput.BeginRead(arrByte,
                        0,
                        arrByte.Length,
                        callbackOnFinish,
                        null);
21     for(long i=0;i<=1000000000;i++){
22         // just to introduce some delay
23     }

24     public static void Main() {
25         FileReader asyncTest=new FileReader();
26         asyncTest.readFile();
27     }
28 }

```

LINE 02

فضاي نام System.IO را به برنامه ملحق مي کند. اين فضاي نام به صورت خودکار حاوي تعريف delegate زير مي باشد:

```
public delegate void AsyncCallback (IAsyncResult ar);
```

LINE 03-10

تعريف کلاس

LINE 06

شيء delegate را تعريف مي کند.

LINE 07-10

سازنده ي کلاس را پياده سازي مي کنند. در اینجا ما تصميم گرفته ايم که بافري حاوي ۲۵۶ بایت را در هر لحظه بخوانيم.

LINE 09

شيء delegate نمونه سازي شده است.

LINE 18-23

readFile را پياده سازي مي کند.

LINE 12-16

نحوه ی استفاده از شیء IAsyncResult را بیان می کند.

LINE 12

sInput.EndRead(result) تعداد بایتهای خوانده شده را بر می گرداند. این خواندن تاجایی که تعداد بایتهای خوانده شده صفر است ادامه پیدا می کند و در اینجا 'Finished reading.' اعلام می گردد.

مقدمه ای بر سی شارپ : قسمت- ۲۲

Delegates and Events

Delegates

Delegates از نوع های مرجع به شمار می آیند که اجازه ی فراخوانی غیر مستقیم توابع را میسر می کنند. نمونه ی ایجاد شده از Delegates ریفرنسی را از چندین تابع در خود نگه می دارد و با فراخوانی يك Delegate تمام این توابع اجرا می گردند. و همانطور که در قسمت های پیشین نیز ذکر شد این مفهوم معادل function pointers در C++ می باشد. در اینجا دو نکته را باید به خاطر سپرد:

۱- Delegates از نوع های مرجع به شمار می آیند و نه نوع های عددی. (نوع های ریفرنس مانند رشته ها و اشیاء ...)

۲- يك Delegate می تواند ارجاعی از چندین متد را در خودش نگه دارد.

تعریف و مقدار دهی اولیه ی Delegates :

يك Delegate را می توان در فضای نام خاص خودش و یا در يك کلاس تعریف نمود. در هر حالت این تعریف از System.MulticastDelegate مشتق شده است. هر delegate به نگهداری مرجعی از توابع با نوعی ویژه محدود می گردد. مثال زیر را در نظر بگیرید:

```
public delegate void Print (String s);
```

از این delegate برای ارجاع به توابعی که تنها دارای يك پارامتر ورودی از نوع رشته و بدون هیچگونه خروجی می باشند ، استفاده می شود. برای مثال فرض کنید که کلاسی حاوی متد زیر باشد:

```
1. public void realMethod (String myString)
2. {
3.     // method code
4. }
```

متدی دیگر در این کلاس می تواند 'Print' delegate را به شکل زیر نمونه سازی کند:

```
Print delegateVariable = new Print(realMethod);
```

که در حقیقت مرجعی از 'realMethod' را در خود نگه می دارد.

نکته : 'multicast delegates'

این نوع delegates می توانند همزمان به متدهای مختلفی ارجاع نمایند. این نوع ها حتما باید خروجی void داشته باشند. برای کار با multicast delegates می توان از عملگر های + و یا - نیز برای اضافه و یا کم نمودن مراجع به آنها استفاده نمود. برای مثال:

1. Print s = null;
2. s = s + new Print (realMethod);
3. s += new Print (otherRealMethod);

مثال زیر روش استفاده از delegates را در عمل بیان می کند:

1. using System;
2. using System.IO;
- 3.
4. public class DelegateTest
5. {
6. public delegate void Print (String s);
- 7.
8. public static void Main()
9. {
10. Print s = new Print (toConsole);
11. Print v = new Print (toFile);

```
12.         Display (s);
13.         Display (v);
14.     }
15.
16.     public static void toConsole (String str)
17.     {
18.         Console.WriteLine(str);
19.     }
20.
21.     public static void toFile (String s)
22.     {
23.         File f = new File("fred.txt");
24.         StreamWriter fileOut = f.CreateText();
25.         fileOut.WriteLine(s);
26.         fileOut.Flush();
27.         fileOut.Close();
28.     }
29.
30.     public static void Display(Print pMethod)
31.     {
32.         pMethod("This should be displayed in the console");
33.     }
34. }
```

در متد Main مثال فوق ، Print delegate دوبار نمونه سازی شده و پارامترهای متفاوتی را پذیرفته است. سپس این delegate ها به تابع Display پاس شده اند.

Events

مثالی ساده از بحث رویدادها و رویداد گردانی کلیک کردن کاربر بر روی یک دکمه و سپس نتیجه ی آن بکار افتادن چندین متد برای مدیریت این آن می باشد. تفاوت متدهای رویداد با متدهای معمولی در این است که آنها باید به صورت خارجی فراخوانی شوند. هر گونه تغییر داخلی در وضعیت برنامه می تواند بعنوان یک رخداد در نظر گرفته شود.

رویداد ها از مدل 'subscription-notification' پیروی می کنند. یک کلاس دلخواه باید قادر به subscription در یک رخداد خاص باشد و سپس هنگامی که رخدادی رویداد یک notification را دریافت کند. Delegates و خصوصاً multicast delegates ، مدل فوق را عینیت می بخشند.

مثال زیر نشان می دهد که چگونه کلاس ۲ در رخداد صادر شده از طرف کلاس ۱ آبونه میشود:

- ۱- کلاس یک صادر کننده ی رخدادهای E می باشد. این کلاس حاوی public multicast delegate D است.
- ۲- کلاس دو توسط متد رویداد گردان M می خواهد به این رویداد عکس العمل نشان دهد. بنابراین به D ریفرنسی از M را اضافه می نماید.
- ۳- کلاس یک هنگامی که خواست رخداد E را صادر کند تنها کافی است که متد D را فراخوانی نماید. این امر سبب می شود تمام متدهای آبونه شده در رخداد E ، فراخوانی گردند.

از واژه ی کلیدی 'event' برای تعریف multicast delegates استفاده می گردد . به مثال زیر دقت نمایید:

1. public class EventIssuer
2. {
3. public delegate void EventDelegate(object from, EventArgs args);
4. public event EventDelegate myEvent;
- 5.
6. public void issueEvent(EventArgs args)
7. {
8. myEvent(this, args);
9. }
10. }

در مثال فوق کلاس EventIssuer حاوی تعریف رویداد myEvent است. هنگامیکه متد issueEvent فراخوانی می گردد ، رخداد myEvent نیز فراخوانی می شود. اگر کلاسی توسط ei EventIssuer ، تمایل داشت در این رویداد آبونه شود باید به صورت زیر عمل نماید:

```
ei.myEvent += new EventIssuer.EventDelegate(handleEvents);
```

مرجع این قسمت :

راهنمای همراه Borland C# builder قسمت Softsteel Solutions C# Tutorial .

مقدمه ای بر سی شارپ : قسمت- ۲۳

مباحث تکمیلی رخدادها (Events) :

رخداد مکانیزمی است که توسط آن یک کلاس می تواند کلاینت های خودش را از اتفاق افتادن امری باخبر سازد. رخدادها توسط Delegates تعریف می شوند.

برای توضیحات بیشتر بهتر است در ابتدا یک برنامه ی کامل را ملاحظه نمود . سپس قسمت به قسمت آن آنالیز خواهد گشت:

```
using System;

public delegate void DivBySevenHandler(object o, DivBySevenEventArgs e);

public class DivBySevenEventArgs : EventArgs
{
    public readonly int TheNumber;

    public DivBySevenEventArgs(int num)
    {
        TheNumber = num;
    }
}

public class DivBySevenListener
{
    public void ShowOnScreen(object o, DivBySevenEventArgs e)
    {
        Console.WriteLine(
            "divisible by seven event raised!!! the guilty party is {0}",
            e.TheNumber);
    }
}

public class BusterBoy
{
    public static event DivBySevenHandler EventSeven;

    public static void Main()
```

```

    {
        DivBySevenListener dbsl = new DivBySevenListener();
        EventSeven += new DivBySevenHandler(dbsl.ShowOnScreen);
        GenNumbers();
    }

    public static void OnEventSeven(DivBySevenEventArgs e)
    {
        if(EventSeven!=null)
            EventSeven(new object(),e);
    }

    public static void GenNumbers()
    {
        for(int i=0;i<99;i++)
        {
            if(i%7==0)
            {
                DivBySevenEventArgs e1 = new DivBySevenEventArgs(i);
                OnEventSeven(e1);
            }
        }
    }
}

```

OUTPUT

```

F:\c#\events>csc 1.cs
Microsoft (R) Visual C# Compiler Version 7.00.9254 [CLR version v1.0.2914]
Copyright (C) Microsoft Corp 2000-2001. All rights reserved.

```

```

F:\c#\events>1
divisible by seven event raised!!! the guilty party is 0
divisible by seven event raised!!! the guilty party is 7
divisible by seven event raised!!! the guilty party is 14
divisible by seven event raised!!! the guilty party is 21
divisible by seven event raised!!! the guilty party is 28
divisible by seven event raised!!! the guilty party is 35
divisible by seven event raised!!! the guilty party is 42
divisible by seven event raised!!! the guilty party is 49
divisible by seven event raised!!! the guilty party is 56
divisible by seven event raised!!! the guilty party is 63
divisible by seven event raised!!! the guilty party is 70
divisible by seven event raised!!! the guilty party is 77
divisible by seven event raised!!! the guilty party is 84
divisible by seven event raised!!! the guilty party is 91
divisible by seven event raised!!! the guilty party is 98

```

```

F:\c#\events>

```

توضيحاتي در مورد كد فوق:

در کد فوق هرگاه عددی تولید شود که بر ۷ قابل قسمت است ، رخدادی صادر می گردد. رویداد گردان (event handler) در این حالت پیغامی را مبتنی بر اتفاق افتادن رخداد بر روی صفحه به همراه عدد مربوطه نمایش می دهد.

اولین کاری که برای این منظور انجام شد تعریف یک delegate است :

```
public delegate void DivBySevenHandler(object o, DivBySevenEventArgs e);
```

این delegate پارامترهایی را که باید به رویداد گردان فرستاد ، تعریف می نماید. بنابراین هر کلاسی که بخواهد این رخداد را اداره نماید باید متدی تعریف کند که دقیقاً آرگومانهای ورودی و خروجی آن همانند این delegate باشد.

همانطور که ملاحظه می نمایید اولین آرگومان Delegate تعریف شده از نوع object می باشد. در مثالهایی واقعی تر و کاربردی تر عموماً تنها یک مرجع از این شیء مورد استفاده قرار می گیرد. هر چند در مثال ما تنها یک new object بعنوان آرگومان به رویداد گردان فرستاده شده است. از رفرنس this نیز می توان در این حالت استفاده نمود. پارامتر دوم از System.EventArgs مشتق شده است. System.EventArgs کلاسی است پایه برای کپسوله کردن داده های مرتبط با رخدادها. ما از آن برای فرستادن اطلاعات مرتبط با رخداد به رویداد گردان استفاده می نماییم.

در ادامه کلاسی مشتق شده از EventArgs را ایجاد می نماییم:

```
public class DivBySevenEventArgs : EventArgs
{
    public readonly int TheNumber;

    public DivBySevenEventArgs(int num)
    {
        TheNumber = num;
    }
}
```

متغیر read-only این کلاس برای نگهداری عدد تولید شده ی قابل قسمت بر هفت مورد استفاده قرار می گیرد. بجای اینکار از properties نیز می توان استفاده کرد.

سپس یک کلاس listener را برای گوش فرا دادن به رخدادهای اتفاق افتاده ایجاد می نماییم:

```
public class DivBySevenListener
{
    public void ShowOnScreen(object o, DivBySevenEventArgs e)
    {
        Console.WriteLine(
            "divisible by seven event raised!!! the guilty party is {0}",
            e.TheNumber);
    }
}
```

این کلاس حاوی متد ShowOnScreen می باشد که با نحوه ی تعریف Delegate برنامه در ابتدای ایجاد آن ، همخوانی دارد. به نحوه ی استفاده از DivBySevenEventArgs برای نمایش عدد قابل قسمت بر هفت دقت نمایید.

در ادامه به آنالیز کلاس حاوی متد Main می پردازیم:

در این کلاس ، رخداد ذکر شده به صورت زیر تعریف می گردد :

```
public static event DivBySevenHandler EventSeven;
```

متد زیر رخداد را دریافت نموده و سپس تمام کلاینت ها را آگاه می سازد :

```
public static void OnEventSeven(DivBySevenEventArgs e)
{
    if (EventSeven != null)
        EventSeven(new object(), e);
}
```

در تابع GenNumbers کدهای درون حلقه، ۹۹ بار اجرا خواهند شد و در هر بار، عملیات بررسی قابل تقسیم بر هفت بودن، انجام می شود.

```
public static void GenNumbers()
{
    for (int i = 0; i < 99; i++)
    {
        if (i % 7 == 0)
        {
            DivBySevenEventArgs e1 = new DivBySevenEventArgs(i);
            OnEventSeven(e1);
        }
    }
}
```

در متد Main ابتدا شیء DivBySevenListener ایجاد می شود. سپس از عملگر += برای اضافه کردن delegate به رخداد استفاده گردیده است. بدیهی است که برای حذف می توان از عملگر -= استفاده کرد.

```
public static void Main()
{
    DivBySevenListener dbsl = new DivBySevenListener();
    EventSeven += new DivBySevenHandler(dbsl.ShowOnScreen);
    GenNumbers();
}
```

پس از انجام این عملیات، تابع GenNumbers فراخوانی می شود. این تابع اعدادی از ۰ تا ۹۸ را تولید می نماید. هرگاه که یکی از اعداد تولیدی بر هفت قابل قسمت باشد خبرش سریعاً به اطلاع عموم خواهد رسید!

نکته :

رخدادها تنها در کلاسی که آنها را تعریف نموده اید قابل استفاده می باشند. این مورد می تواند مشکلاتی را در بحث ارث بری ایجاد نماید. برای حل این مساله می توان متد رخداد را protected معرفی نمود تا کلاس ارث برده از کلاس اصلی بتواند آنرا فراخوانی نماید و همچنین بهتر است آنرا virtual نیز معرفی کرد تا بتوان آنرا override (تعریف) نمود.

به صورت خلاصه برای بکار گیری رخدادها باید مراحل زیر طی شود:

- ۱- تعریف يك Delegate
- ۲- تعریف کلاسی مشتق شده از System.EventArgs برای کپسوله کردن اطلاعات مربوط به رخداد.
- ۳- ایجاد کلاسی که رخداد را برانگیزد (raising an event). این کلاس شامل موارد زیر خواهد بود: الف) رخدادی که دارای کلاینت های رجیستر شده است. ب) متدی برای آگاه ساختن کلاینت ها از وقوع يك رخداد

- ۴- ایجاد کلاس کلاینت. این کلاس دارای متدی است که با امضای delegate همخوانی دارد (نوع و تعداد آرگومانهای ورودی و خروجی آنها یکی است). این متد رخداد را دریافت خواهد کرد.
- ۵- رجیستر کردن این متد بعنوان یکی از گوش دهندگان مجاز به استفاده از رخداد.

یک مثال کامل دیگر برای علاقمندان :

```
//Start of program
using System;
using System.ComponentModel;

// First step -- declare the delegate
public delegate void EvenEventHandler(object sender, EvenEventArgs e);

//Second step -- create a class derived from EventArgs
public class EvenEventArgs : EventArgs
{
    // Declare private variables to reflect the information
    // about the event
    private readonly bool evensteven;
    private readonly int evenone, eventtwo;

    //Constructor
    public EvenEventArgs(bool evensteven, int evenone, int eventtwo)
    {
        this.evensteven = evensteven;
        this.evenone = evenone;
        this.eventtwo = eventtwo;
    }

    //The properties to enable the client to access the event info
    public bool Evensteven
    {
        get
        {
            return evensteven;
        }
    }

    public int Evenone
    {
        get
        {
            return evenone;
        }
    }

    public int Eventtwo
    {
        get
        {
            return eventtwo;
        }
    }
}
```

```
//Third step -- create the class that will raise the event
public class EvenDetector
{
    //Declare some variables to make life simpler
    private bool goteven, done;
    //and our own random number generator
    private Random r1;
    //as well as the number we shall check for 'evenness'
    private int randomnum;
    //Also the two even numbers
    private int evenone, eventwo;

    //Constructor
    public EvenDetector()
    {
        r1 = new Random();
    }

    public void numbercruncher()
    {
        //Loop until two successive even numbers have been generated
        while(!done)
        {
            randomnum = (int)(100*r1.NextDouble());

            if(randomnum%2==0)
            {
                if(goteven)
                {
                    done =true;
                    eventwo = randomnum;
                }
                else
                {
                    goteven = true;
                    evenone = randomnum;
                }
            }
            else
            {
                goteven = false;
            }
        }

        //Success -- create an object for the client(s)
        EvenEventArgs e = new EvenEventArgs(done, evenone, eventwo);

        //and call the method to send it to the client(s)
        OnEven(e);
    }

    //Step 3a -- the event is declared
    public event EventHandler Even;

    //Step 3b -- the method that notifies client(s)
    protected virtual void OnEven(EventArgs e)
    {
        // If the list of clients is NOT empty
        if(Even != null)
    }
```

```

    {
        //despatch the event to each client
        Even(this, e);
    }
}

//Fourth step -- and now the client class
public class EvenListener
{
    //This is the method with the same signature as
    //the delegate. It will receive the event
    public void EvenAnnouncer(object sender, EvenEventArgs e)
    {
        //This will always be true and hence is redundant.
        //Just illustrates the use of EventArgs
        if(e.Evensteven)
        {
            Console.WriteLine("THE EVEN TWINS ARE HERE! -- {0} and {1}", e.Evenone,
e.Eventtwo);
        }
    }
}

//Fifth step -- hook 'em up
public class EvenTester
{
    public static void Main()
    {
        //Instantiate EvenDetector
        EvenDetector ed = new EvenDetector();

        //Instantiate EvenListener
        EvenListener el = new EvenListener();

        //Register the listener method
        ed.Even += new EvenEventHandler(el.EvenAnnouncer);

        ed.numbercruncher();
    }
}

//End of program

```

مراجع :

Events and event handling in C# By Nish (<http://www.codeproject.com/>)
 Handling Events In C# By Biswajit Sarkar (<http://www.csharp4help.com/>)

مقدمه ای بر سی شارپ : قسمت- ۲۴

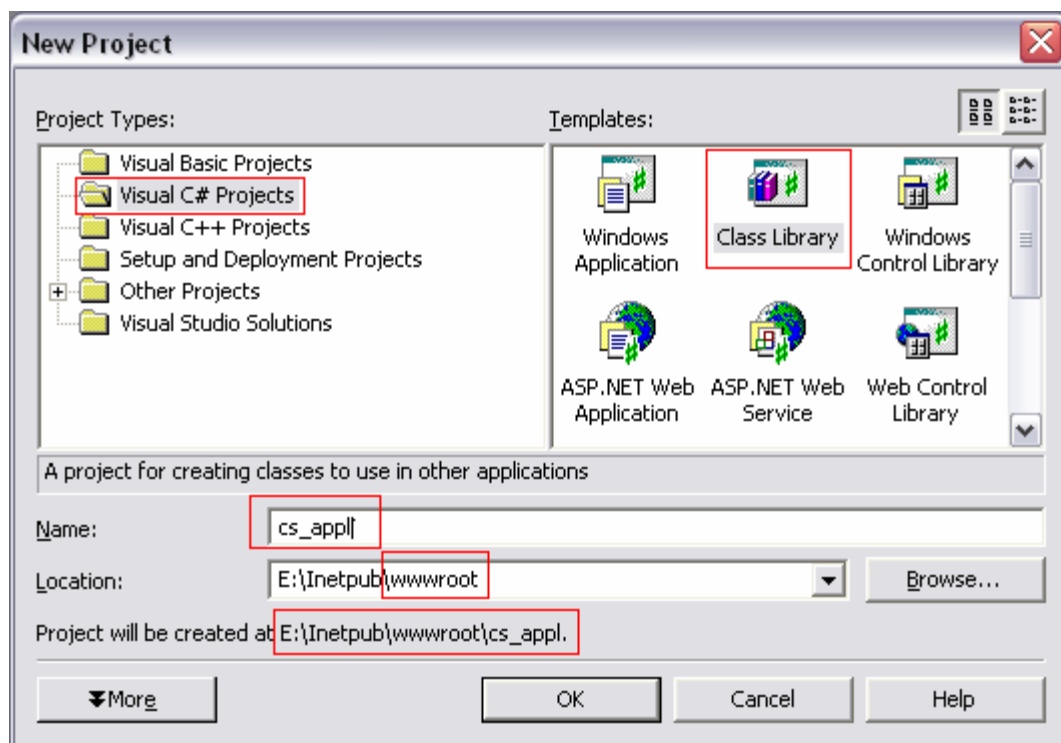
اپلت های سی شارپ

مقدمه :

در این مقاله قصد داریم با نحوه ی نوشتن اپلت های سی شارپ آشنا شویم. لازم به ذکر است که اپلت های سی شارپ تنها با استفاده از IIS قابل اجرا هستند (برخلاف اپلت های جاوا). بنابراین اگر در محیطی غیر از این محیط (IIS) سعی به اجرای آنها نمایید صرفاً با یک صفحه ی خالی مواجه خواهید شد. برخلاف برنامه های ASP.NET که در محیط IIS حتماً باید توسط یک دایرکتوری مجازی ایزوله شوند در مورد این اپلت ها صرفاً کپی کردن فولدر به درون wwwroot کفایت می نماید. برای اجرای اپلت های سی شارپ کامپیوتر های کلاینت حداقل نیاز به IE6 و دات نت فریم ورک دارند.

ایجاد اپلتهای ساده با استفاده از سی شارپ :

VS.NET را اجرا نموده و سپس نوع پروژه را VC# و از پنل Templates گزینه ی Class Library را انتخاب نمایید. در اینجا قصد داریم کد اپلت را به صورت یک اسمبلی (.dll) در آورده و در صفحات HTML از آن استفاده نماییم (شکل یک).



شکل ۱- ایجاد یک پروژه ی Control library درون wwwroot برای ایجاد یک سی شارپ اپلت.

لازم به ذکر است که سازنده ي کلاس پروژه ي ما بايد بدون پارامتر باشد تا توسط IE قابل فراخواني باشد. پس از ساخت اين کلاس و توليد اسمبلي آن به صورت زیر به سادگي مي توان از آن در صفحات HTML استفاده نمود:

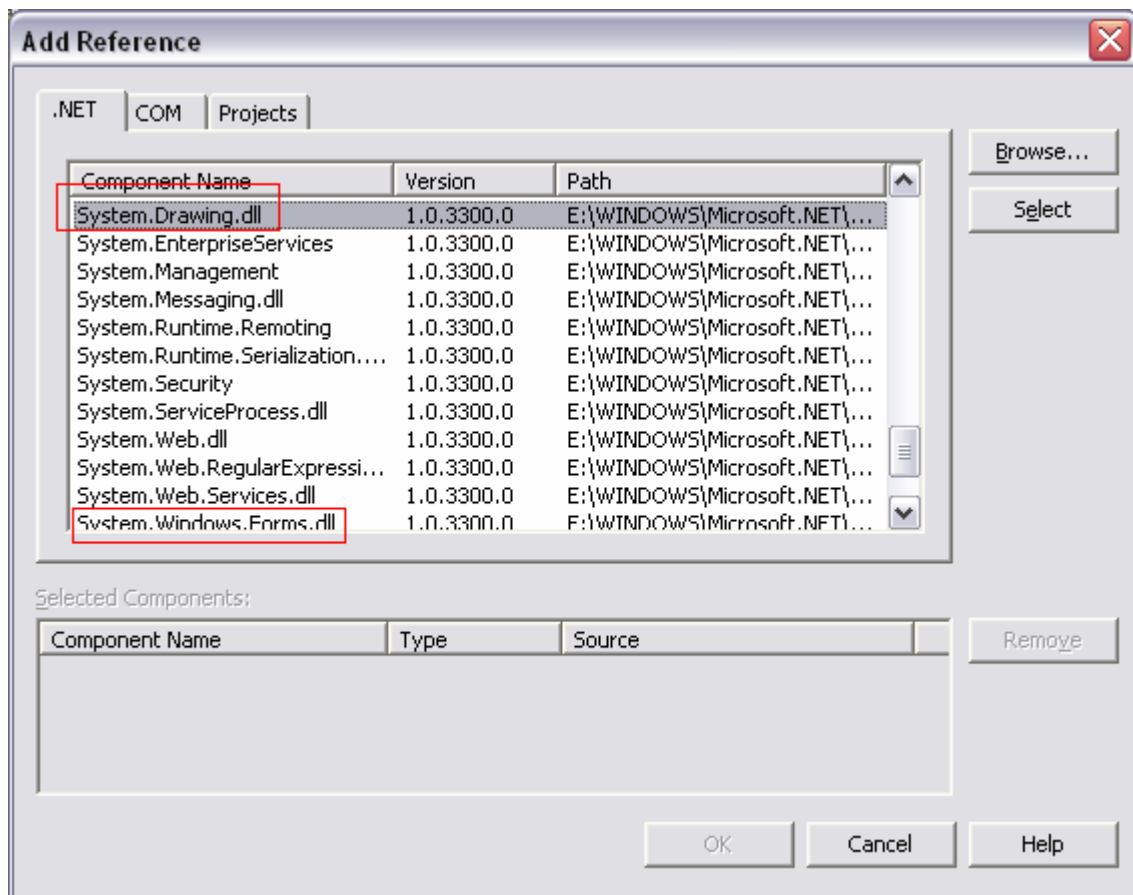
```
<object
  id=anID
  classid="http:myAssemblyDll.dll#controlClassToInstantiate"
  height=300 width=300>
</object>
```

پس از نسبت دادن ID لازم به کنترل مي توان به صورت زیر آنرا درون کد يك جاوا اسکریپت فراخواني نمود:

```
<script> <!--
  myID.AProperty = aValue;
  myID.Refresh()
//--></script>
```

کد اپلت :

سورس کامل اسمبلي در ذیل ارائه شده است. تنها بايد بخاطر داشت که قبل از کامپایل کردن برنامه، ریفرنس هاي لازمي را که در شکل دو مشخص شده اند را بايد به پروژه ضميمه نمود.



شکل ۲- اضافه نمودن ریفرنس های لازم به پروژه ی اسمبلی دات نت.

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace cs_appl
{
    public class T : Control
    {
        protected override void OnPaint(PaintEventArgs e)
        {
            e.Graphics.FillRectangle(
                new SolidBrush(Color.Azure), ClientRectangle);
            e.Graphics.DrawLine(Pens.DarkSalmon,
                new Point(0, 0),
                new Point(
                    ClientRectangle.Width,
                    ClientRectangle.Height));
        }
        public static void Main(string[] m)
    }
}
```

```

    {
        Form f = new Form();
        T t = new T();
        t.Dock = DockStyle.Fill;
        f.Controls.Add(t);
        Application.Run(f);
    }
}

```

کد کامل صفحه ي HTML ابي که از کتابخانه ي ساخته شده ي فوق مي تواند استفاده نمايد به صورت زیر است:

```

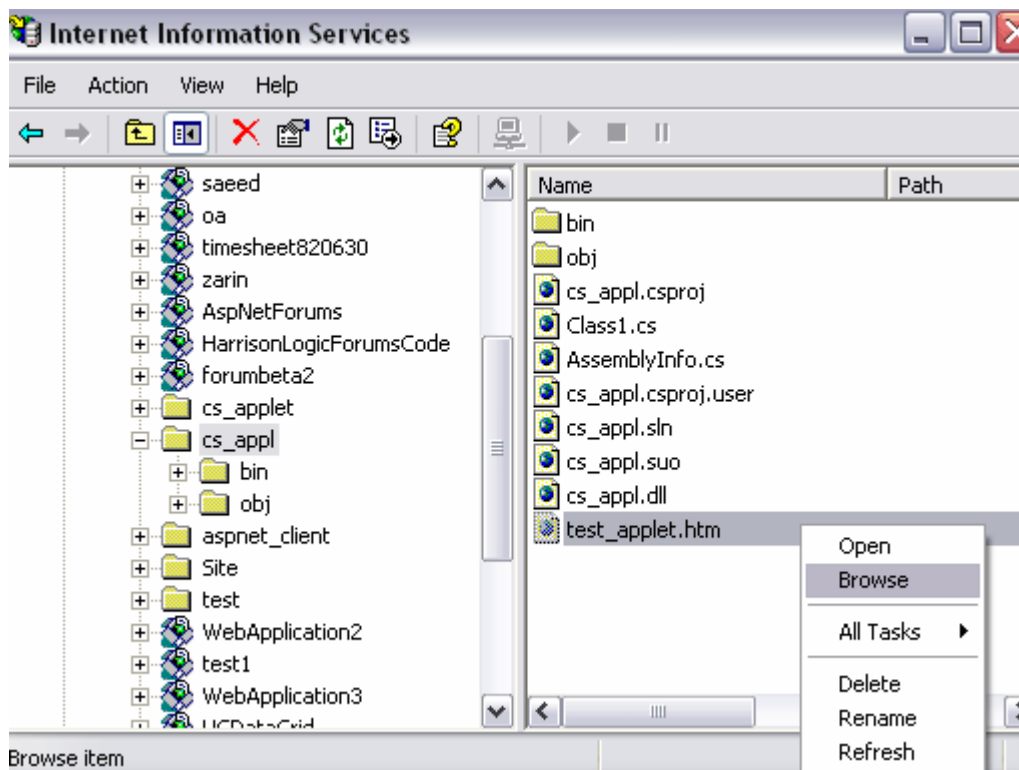
<html>
<head>
<title>C# Web control I</title>
</head>
<body>
and now...<br>
<object id=t
classid="http:cs_appl.dll#cs_appl.T"
height=300 width=300 VIEWASTEXT>
</object>

<br>
<a href=Class1.cs>source</a>
</body>
</html>

```

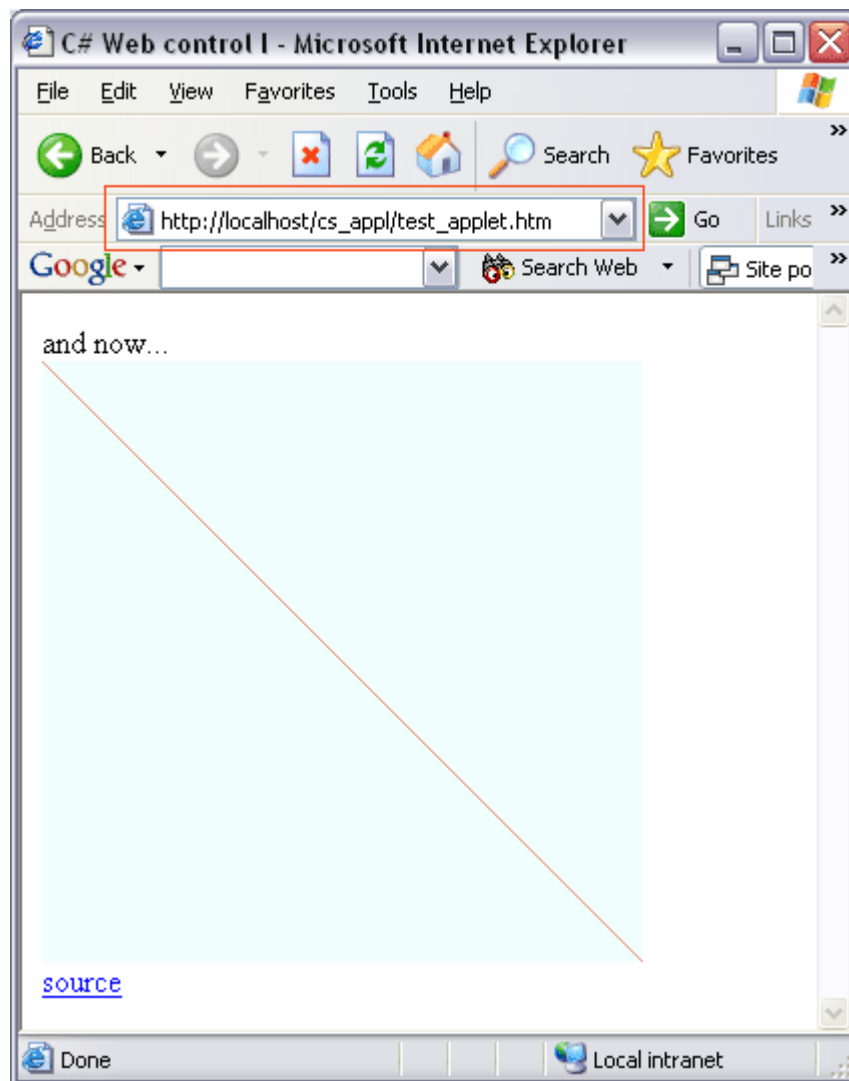
نحوه ي اجرا کردن اپلت و HTML مربوطه در IIS :

هر چند الزامي به ساخت پروژه درون wwwroot وجود نداشت اما هنگام اجراي اپلت سي شارپ بايد آنرا به درون wwwroot کپی نمود. روي فایل HTML ساخته شده در محيط IIS کليک راست نموده و گزینه ي Browse را برگزينيد. (شکل ۳)



شکل ۳- نحوه ی اجرای اپلت سی شارپ در IIS .

و در پایان اپلت به صورت شکل ۴ اجرا خواهد شد.



شكل ٤- نمايي از اجراي يك اپلت نمونه ي سي شارپ.

مرجع :

C# Applet By Lloyd Dupont
<http://www.csharp-help.com/>